Crimson 2

使用手册

中文 6.0 版

2005-6-28

red lion

红狮工控 2003 年出版

此手册由 Mike Granby 和 Jesse Benefiel 编写。

6.0 版中文手册是根据 6.0 版英文手册翻译的。

内容索引

Crimson 2

使用手册

入门

欢迎使用 Crimson 2 软件----这是由红狮工控提供的最新系列人机界面的编程软件，Crimson 提供一个快速简单的方法来操作 G3 系列人机界面的各种功能，同时又可使资深用户运用高端性能，如：Crimson 独特的语言编程平台。

系统要求

Crimson 2 对 PC 机的硬件要求：

- 奔腾级处理器按操作系统的要求
- 内存和硬盘空间按系统要求
- 约 10MB 的硬盘空间供软件安装使用
- 显示器分辨率最低 800 X 600，建议使用 256 色以上。
- 一个 RS232 口或 USB 口供下载程序至 G3 人机界面。

Crimson 2 可运行在 Windows 95 以上版本的系统。如用户希望使用 G3 人机界面提供的 USB 口，则必须使用 Windows 98 以上版本。如用户希望使用 USB 口远程访问 G3 人机界面的 CF 卡，我们建议用户使用 Windows 2000 或 Windows XP。虽然 Windows 98 系统具备访问 CF 卡的功能，但较新版本的系统会提供更全面的功能，例如：当用户选择锁定 CF 卡时，可防止 G3 对 CF 卡的写操作，缩短运行时间。

安装软件：

如果你从红狮网站下载了 Crimson 2 安装软件，直接执行所下载的文件，按指令安装。如果你使用光碟版的 Crimson 2 软件，请将安装盘放置在 CDROM 驱动器中，按指令执行。如果指令没有出现，你可能没有选择自动运行功能，你可在开始菜单的运行选项键入 x:\setup，回车后按指令安装，这里 x 指你计算机中的 CDROM 驱动器名。

版本更新

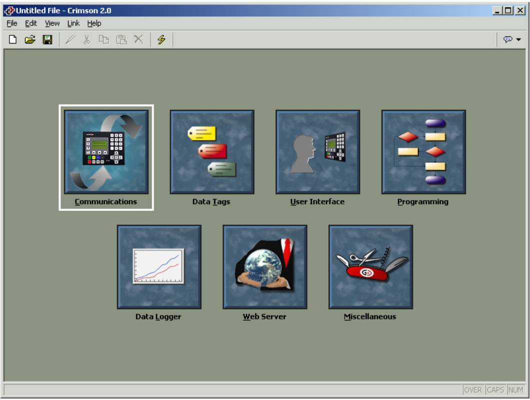
如果你已接入互联网，可运行 HELP 菜单中的 check for update....，Crimson 2 会访问红狮公司网站，如果有更新版本的软件，将会自动下载和安装。

安装 USB 驱动程序：

当你的电脑第一次使用 USB 口接入 G3 人机界面，Windows 会提示你选择设备的驱动程序，缺省的驱动程序位置是 c:\Program File\Red Lion Controls\Crimson 2.0\Device。当硬件安装精灵出现，选择浏览选项，将安装精灵指向缺省位置或其他在安装软件时你指定的位置。正确地选择这一步是非常重要的，否则你必须在设备管理器中手动删除驱动程序，再重新安装。

Crimson 软件基础

启动 Crimson 软件，请从开始菜单\程序\红狮控制文件夹\选择 Crimson 图标，Crimson 主页面出现了，如下出现的图标是用来设置人机界面的不同功能和特性....



主页面中的前三个图标是最常用的基本功能应用，其余的图标提供了 G3 的高级功能，如编程、数据记录和网络服务器功能。

主页面图标：

本节将依次浏览各图标的基本功能：

通讯：



这个图标是用来设定 G3 的串行通讯口和以太网口的通讯协议。当通讯口使用主站协议时（如：G3 使用此协议向远程设备传输数据并从设备获取数据反馈），用户可使用此图标设置一个或多个可以被访问的设备。当使用从站设备时（如：G3 使用此协议接收远端设备或系统的数据命令并反馈相应数据），用户可用此图标规定哪些内部数据可被读取访问。用户也可使用此图标完成远端设备间的数据交换，也就是协议转换功能。

数据变量：



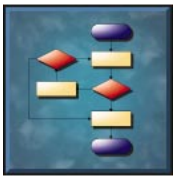
这个图标是用来设定可被远程设备访问的数据，以及用于存储内部信息的数据。每个数据都规定了相应的各种属性，最基本的属性就是数据格式，它是用来设定数据变量中的数据是以怎样形式显示在人机界面或网页上的。当完成数据变量的定义后，无论在任何地方显示，用户都不必重复去设置这些参数了。更高级的数据变量属性还包括：报警---在数据变量指定的一些状态出现时发出；触发---在指定的状态出现时，执行相应的可编程的动作。

用户界面



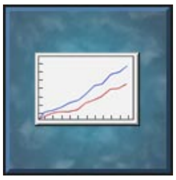
此图标用于创建和编辑显示的页面，并设定人机界面的各按键的功能。页面编辑器可以使用户显示各种的图形，我们称之为元件。无论是简单的图形，如线条和矩形，或者是复杂的图形，都可以与特别的变量和表达式相联系。通常这些元件使用规定的信息，这些信息是在变量创建时定义的，但这些信息是可以按需要随时更改。

编程



这个图标用来创建和编辑程序。它是用 Crimson 2 特有的类 C 语言来编写的。这些程序可以运用系统中的数据来进行复杂的判断和数据运算，它可以充分地扩展 Crimson 的功能，以保证在最复杂的应用要求下，也可以轻松地处理。

数据记录



这个图标用来创建和编辑数据记录。它可以向 G3 的 CF 卡存储任意数量的数据，数据最快可以每秒记录一次。数据是以 CSV 格式存储的，它可以轻松地输入到各种应用程序中，如：Microsoft Excel。卡中的数据可以用多种方式读取，可直接取出 CF 卡，或使用 G3 的 USB 接口与 PC 相连来读取，或使用 Crimson 的 web server 功能，通过以太网口来读取。

网络服务器：



这个图标用来设置 Crimson 的网络服务器功能，及创建和编辑网页。网络服务器功能可以提供用多种途径远程访问 G3。首先用户可以使用 Crimson 的一个自动生成网页，网页可以包含各种变量列表，每个变量均按照已规定的变量属性定义。其次用户可以使用第三方软件，如 Microsoft Frontpage，来设计一个自定义的网页，并使用一些特殊字符来命令 Crimson 插入动态变量。同时，用户可以使用 Crimson2 独特地远程访问和控制功能，它可以使用网络浏览器来浏览 G3 的显示和控制它的按键。网络服务器还可以用来访问数据记录器中的 CSV 文件。

杂项：

这是个保留图标，将来用来扩展功能。

使用即时帮助：

Crimson 提供一个非常实用的功能，称为即时气泡帮助：



这个功能可以使用户浏览各个项目的相关帮助文件，如主页面中的各个图标，对话框和对话窗的各个项目等。它是用工具栏最右边的图标来控制的，可以被设定为三种模式：不显示---气泡帮助功能不启用；鼠标指向时显示---当鼠标指向特定项目并停留时将显示相应帮助；选择时显示---当选择特定项目后显示相应帮助。

运用数据库：

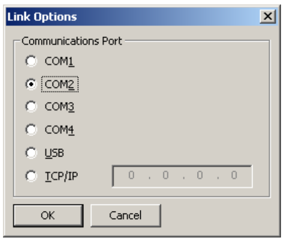
Crimson 将所有关于人机界面的设置，存储在被称作数据库的文件中。此文件是以 CD2 为扩展名，尽管 Windows Explore 可能隐藏其缺省扩展名。Crimson 的数据库文件与以往红狮公司的人机界面的设置文件不同，它是文本型文件，所以在意外损坏时非常容易修复。数据库文件可以通过 File 菜单下的各种命令操作，除 Save Image 命令外，其他命令与 Windows 的标准命令完全相同，无需多加解释，Save Image 命令会在以后介绍。

下载文件至人机界面：

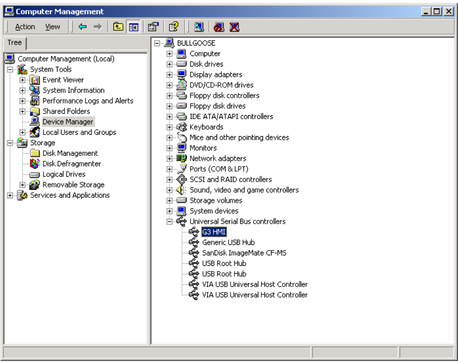
Crimson 的数据库文件是通过 Link 菜单中的命令来下载文件至人机界面。文件的下载通常只需数秒钟，但如果 Crimson 需要更新人机界面的固件配置或该数据库文件是第一次下载到人机界面时，则需要花费较多时间。当第一次下载完成后，Crimson 将使用一种称为增量下载的方式来更新文件，以保证只传输文件的被改动部分。这样数据库文件的更新只需数秒钟即可完成，大大缩短了开发时间和调试过程。

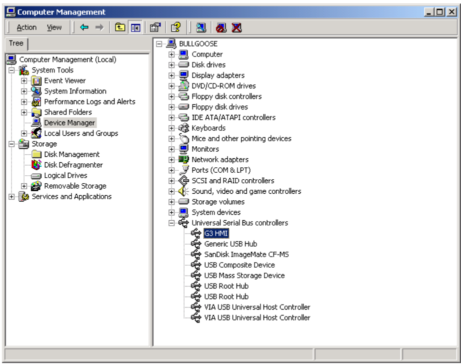
设置连接：

PC 与 G3 人机界面间的编程连接可以通过 RS-232 串行口和 USB 接口两种方式完成。在下载程序前，用户需使用 Link 菜单下的 Option 命令，来选择正确的串行通讯口或可用的 USB 接口。



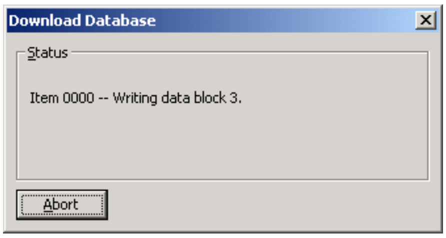
在使用 USB 接口之前，用户应确认 G3 的 USB 驱动程序已正确安装，为确信这一点，将 G3 连接至 PC 的 USB 口，如驱动程序未安装，则根据手册开始时的说明操作，然后打开操作系统中的设备管理器，展开 USB 图标，显示 G3 的图标，确信 G3 的图标上没有任何告警记号。如果有告警，则需删除设备，断开和重新连接 G3，重新安装并验证你已正确地安装完毕。下面的示图分别地展示了典型的设备管理器中闪存卡的卸载和安装。





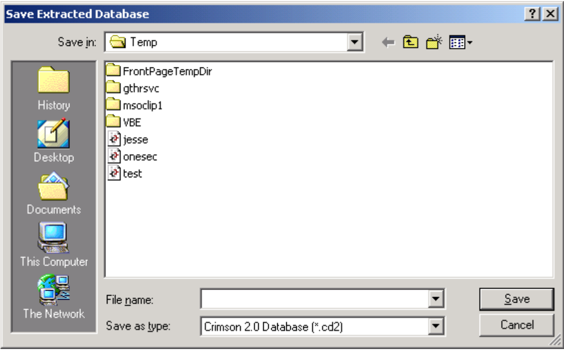
发送数据库文件：

一旦连接设置完成，数据库文件就可以通过 Link-Send 或 Link-Update 指令来下载。Send 指令将传送整个文件，无论文件中的项目是否有更改。Update 指令将仅传送有更改的项目，它会花费较少的时间。当 Update 下载中遇到任何问题，Update 下载方式会自动转向完整传送方式。工具栏中的闪电图标和 F9 键是 Update 指令的快捷方式。



提取数据库文件

Link-Support Upload 指令是用来设定当 Crimson 下载数据库文件至 G3 屏时，是否支持上载功能。支持上载会减慢下载速度，但当你丢失了你的数据库文件，你可以从 G3 屏中提取整个数据库文件。如果你没有设置支持上载功能，并且丢失了数据库文件，你将不得不重新编程。使用 Link-Extract 命令，你可以提取数据库文件，这个指令将上载文件，并提示你输入需存储的文件名，然后将打开文件。



安装闪存卡：

如果你通过 USB 端口连接到 G3 屏，你可以命令 Crimson 安装 G3 中的 COMPACTFLASH 卡作为 Windows Explorer 中的一个驱动器。你可以使用这一功能向卡中存储文件或是读取卡中 Data logger 存储的文件信息。驱动器的安装和卸载是通过 Link 菜单中的 Mount Flash 和 Dismount Flash 指令完成的。一旦指令发出，G3 屏会被复位，Windows 会刷新相应的窗口以显示或隐藏 CompactFlash 驱动器。



注意：当安装 CompactFlash 卡时，会遇到一些告警：

- 当卡安装完毕后，当卡中的数据被更改时 G3 会周期性的通知 PC。这就意味着 PC 和 G3 在数据记录操作中都会花费更长的时间。

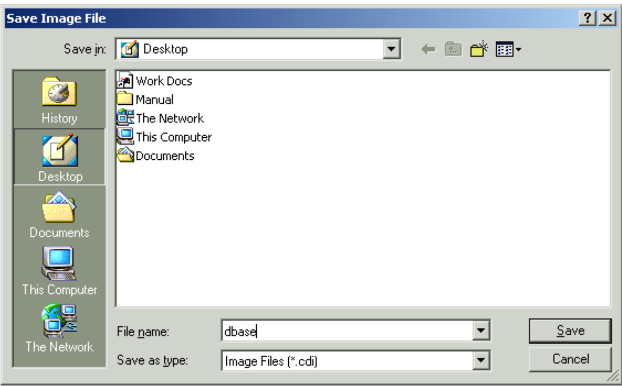
- 如果你从 PC 向卡中写数据，G3 将不能访问 CF 卡，直到 Windows 解除了对 CF 卡内容的锁定。这将会花费一分钟，并限制其间的数据记录操作和对自定义网页的访问。Crimson 会使用 G3 的 RAM 来保证不会丢失数据，但频繁的写操作会使 CF 卡被锁定更长的时间，将会导致数据丢失。特别注意 Windows98 会在不必要的情况下锁定 CF 卡，而 Windows 2000 或 XP 会是比较好的选择。
- 请不要通过 Window Explorer 或 Command 提示符来格式化 CF 卡，因为 Windows 不能正确的锁定 CF 卡，格式化会导致 CF 卡操作不可靠和数据丢失。正确的方法是通过 Crimson 中的 Format Flash 指令或用独立的 CF 卡适配器用 FAT16 文件执行格式化。

传送时间：

Link-Send Time 指令用来将 G3 中的时钟与 PC 中的时钟对时。对时前应先确认你的时钟是正确的。

通过 CF 卡更新数据库文件：

如果人机屏已安装在用户现场，你需要更新其中的数据库文件，Crimson 允许你将数据库文件拷贝至 CF 卡，将卡发送到用户手中，用户可通过卡来直接载入数据库文件。这个过程是通过 File 菜单中的 Save Image 指令来完成的。



Save Image 指令将创建一个以 CDI 为扩展名的 Crimson 数据库文件的印象文件，它同时将当前 G3 的固件信息复制在一个以 BIN 为扩展名的文件中，印象文件必须是以 DBASE.CDI 为文件名，它和 BIN 文件都必须放置在 CF 卡的根目录下，当更新人机屏时，首先断电，插入 CF 卡（存有两个文件），重新上电，G3 的加载引导器首先检查是否需要更新设备固件，当此过程完成后， Crimson 的实时应用程序将加载存储于卡上的数据库文件，整个过程就完成后，CF 卡就可以被卸除或按需要保留。

GURU MEDITATION 代码：

如果 G3 人机屏中的应用程序发生运行错误，将会使人机屏产生复位，而引起错误的状态会被记录。当人机屏重新起动，这个信息会以 GURU MEDITATION 代码的方式来显示。典型的 GURU MEDITATION 码是如下格式：

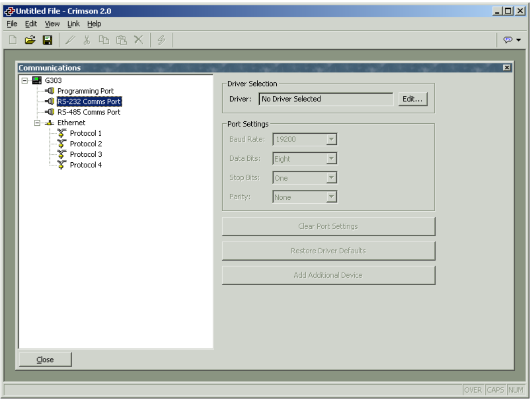
03-2004-1BE4-205

这个信息可以 F1 键来清除，此时人机屏会返回正常操作。注意：当 GMC 码显示时，通讯，数据记录和 Web server 功能将处于正常状态，只有用户页面被中断。系统的中断被最小化，如协议转换等功能会正常工作。

在清除 GMC 代码前，最好能记录下代码。你可以将此 GMC 代码和数据库文件电邮至红狮公司的技术支持，我们可以根据此代码推测出问题的原因。

设置通讯

创建 Crimson 数据库的第一步是设置各通讯口的通讯协议和需访问的远程设备。这个操作是通过通讯窗口来完成的，通讯窗口是在 Crimson 主页面中选择第一个图标来打开。



从上图可以看到，通讯窗口中以树状结构列出了设备的可用通讯口。G3 人机屏具有 3 个基本串行通讯口，在选项中可以用扩展卡增加 2 个通讯口。另外，还提供一个单独的，可同时运行 4 种协议的以太网口。

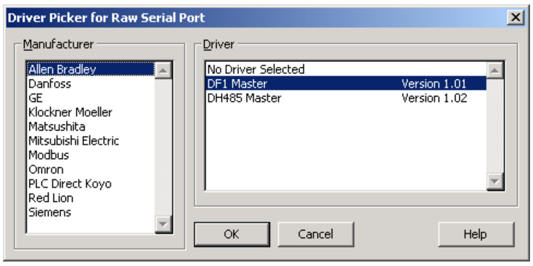
串行口的使用：

当决定使用哪一个串行口用于通讯时，请注意：

- G303 采用一个串行通讯控制器控制它的 RS232 和 RS485 口。这就是说，当任一个通讯口被设置用从站协议，则另一个通讯口将不能再使用。而当一个令牌环协议被使用时，如 Allen-Bradly 的 DH-485，另一个通讯口也同样不能使用。其他 G3 人机界面则没有这个限制。
- 人机界面的编程口可以作为通讯口使用，但此时它就不能被用作下载程序。这不会有影响操作，由于 USB 口可以完成程序下载。当你将串行设备连接至编程口时，特别建议使用 USB 口下载程序。

选择通讯协议：

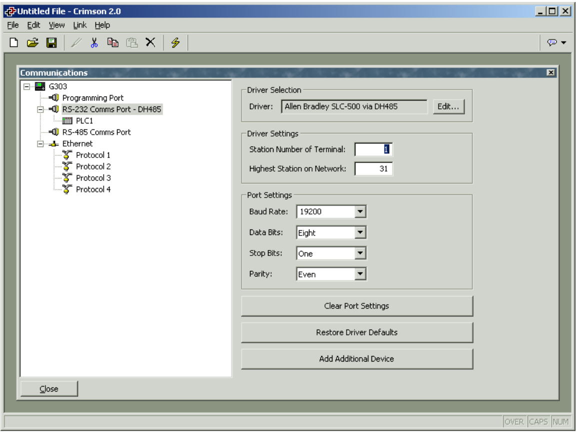
对指定的通讯口设置协议，首先点击通讯窗口中左手栏的通讯口图标，再点击右手栏中驱动器框后面的 EDIT 按钮，将出现下面的对话框：



选择合适的制造商和协议，然后按 OK 键关闭对话框。此时通讯口已设置了相应的协议，一个设备图标出现在左手栏中。当你设置了一个通讯口，多个通讯口参数设置项（波特率，数据位，停止位和奇偶校验），将被设置为协议的缺省值。你可以仔细地检查这些设定，以确保符合设备的要求。

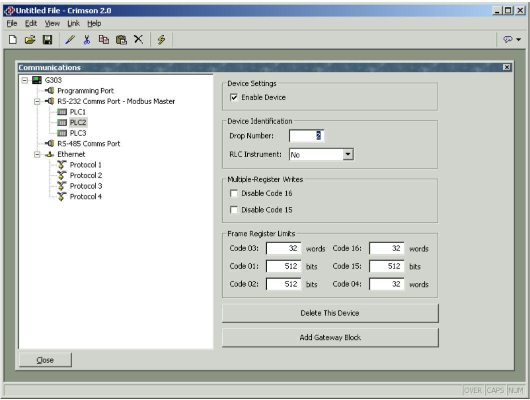
协议选项：

一些协议需要规定附加的参数设置，当对应通讯口被选中时，这些参数将出现在通讯窗口的右手栏中。下面的例子是 Allen-Bradley 的 DH-485 协议中的附加参数，它显示在协议设置框的下方。



连接外部设备：

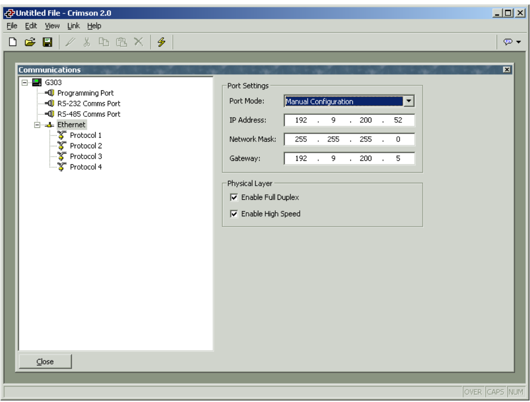
前面提到，当通讯协议选择完毕，会在相应通讯口图标下创建一个设备图标。在使用主站协议时，这个图标代表由协议指定地址的远端设备。如协议支持访问多个设备，你可以用 Add Additional Device 按钮添加设备。每个设备都会由左手栏中的一个图标表示，并且按协议的规定，有一些属性需设置：



如上实例：选择了 Modbus Universal Master protocol，另增加了两个设备，总共有 3 个远端设备可以访问。右手栏中显示了单独设备的属性，Enable Device 属性适用于所有设备和协议，而其它项目是按所选协议的不同而不同。每个设备的名称是由 Crimson 按缺省名字命名的，点击相应图标，键入新的设备名就可更改了。

以太网口设置：

G3 的以太网口是通过通讯窗口中左手栏的 Ethernet 图标来设置的，当这个图标被选中，会显示如下的设置：



IP 地址：

端口模式框用来控制端口的使能和获得 IP 地址的方法。如果选择 DHCP 模式，G3 将尝试从局域网的 DHCP 服务器上获得 IP 地址和相关参数。如果端口被定义为从站协议或支持网页功能，只有当 DHCP 服务器分配一个公开的 IP 地址作为设备的 MAC 地址时，选择 DHCP 模式才能生效，否则用户将无法确定人机界面的地址！

如果选择最常用的 Manual Configuration 模式，那么 IP 地址，网络掩码和网关框都必须输入正确的信息。框中提供的缺省值大多数情况下不适合你的应用！一定要向网络管理员咨询这些参数，正确地输入和下载到 G3 后再连接你的网络。如这些参数有误，将很可能引起网络故障。

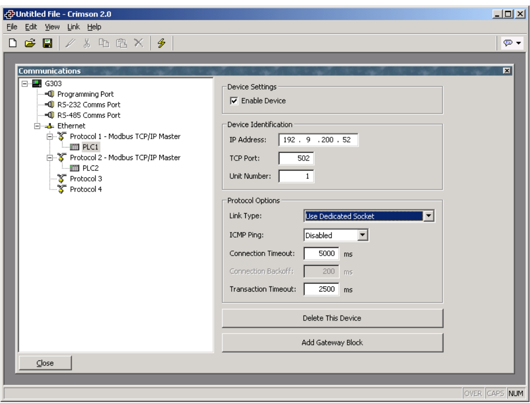
物理层：

物理层选项控制 G3 和与之相连的集线器间数据交换的方式。通常这些选项应保留缺省值，但当网络连接出现问题，特别是 G3 直接与 PC 相连而不通过集线器和交换器，可尝试关闭 Full Duplex 和 High Speed 操作，看是否可解决问题。

协议选择：

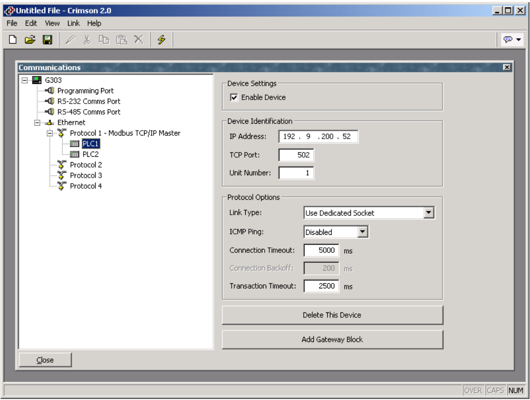
当以太网口设置完成后，你就可以选择你需要使用的通讯协议。你最多可以同时选择 4 个协议，多数协议可支持多个远程设备，这意味着你可以用多种方式来集成不同协议的多个设备间通讯。例如：假设你想要连接两个使用 Modbus TCP/IP 协议的从站设备：

第一种方法是将两个以太网口协议都定义为 Modbus TCP/IP Master，每个协议下只附加一个设备。



对于多数协议，这种方法具有较高的性能，因为它可同时与两个设备通讯。当然它也占据了 4 个协议中的 2 个，限制了你在复杂应用中增加新协议的空间。

第二种方法是定义单独一个 Modbus TCP/IP Master 协议，但在同一个协议下增加多个设备：



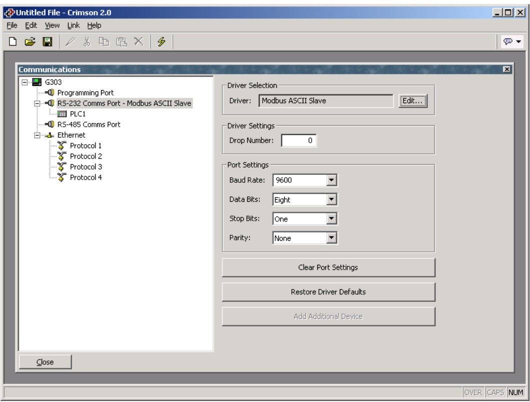
这将稍稍地减慢设备性能，因为 Crimson 会逐个轮询每一个设备，而不是与两个设备同时通讯。但它可节省协议，在复杂应用情况下可最大限度地利用资源。

从站协议：

作为主站协议（是由 G3 来初始化通讯），在通讯图表下没有进一步的设置。而作为从站协议（G3 需要接收和响应远端的请求），这个过程稍稍复杂一些，你必须设置哪些数据可以向远端提供。

选择协议：

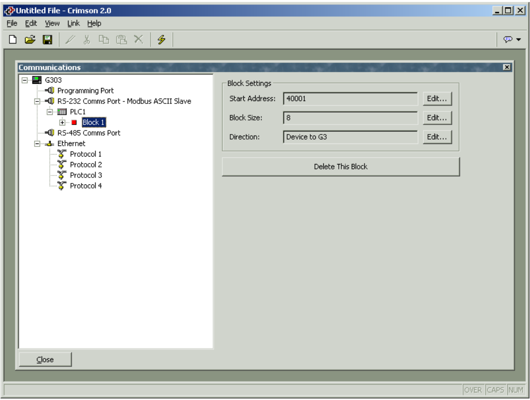
与选择主站协议一样，第一步是在通讯口选择你所需的协议，下面的例子是在 G3 的 RS232 口定义一个 Modbus ASCII 从站协议：



注意一个设备会自动地在协议下创建。在使用主站协议时，它代表 G3 要访问的远端设备。而现在，则代表使用从站协议的 G3，这意味着只有一个独立设备。而象站号等 G3 需要设置的参数，是通过通讯口参数设置的，而不是通过设备参数。

增加网关块：

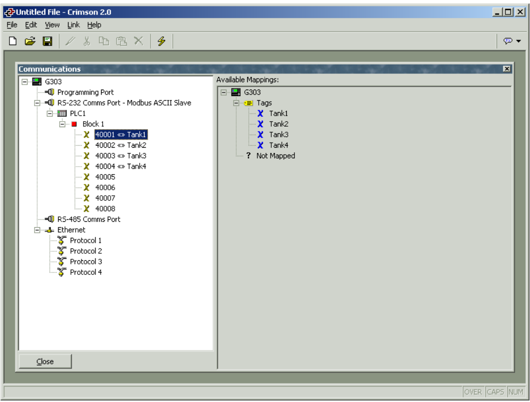
设置完协议后，你必须决定在从站协议下开放的地址范围。例如：我们要使用 Modbus 寄存器 40001 至 40008 来对我们数据库中的一些数据进行读和写访问，我们首先选择通讯窗口左手栏中的设备图标，点击右手栏中的 Add Gateway Block 按钮，一个代表数据块 1 的图标出现了，选中它将会出现如下的设置：



在上面的例子中，我们可以定义起始地址为 40001，这表示块的开始地址。然后我们定义数据块的大小为 8，这样我们就为每一个需要开放的寄存器指定了一个标签。最后我们选择传送方向是从设备到 G3，这样就确定远端设备通过这个数据块可以读/写所开放的数据。

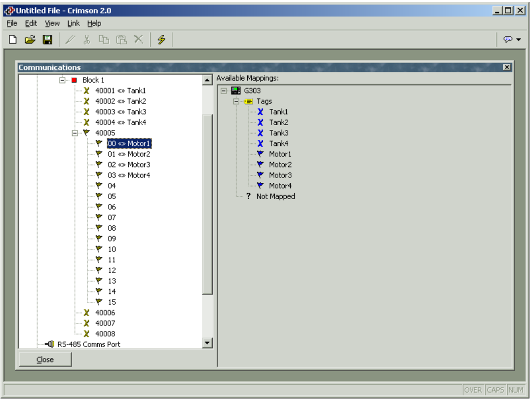
向数据块中添加项目：

当创建了数据块，定义了它的大小，多个项会出现在窗口的左手栏中，它们代表数据块中的每一个寄存器。当选择了其中一个项后，在右手栏中会显示可用的数据项目清单，包含了数据库文件中的数据标签和已定义的每个主站设备中的寄存器，



要指定数据块中的一个寄存器代表一个特定的数据项目，只需简单地拖动右手栏中的数据项目到左手栏中，并放置在数据块中的合适项上即可。上面的例子显示了怎样将数据块中的前 4 个寄存器映射到 Tank1 至 Tank4 的数据标签，这代表对寄存器 40001 到 40004 的访问会被映射至相应的变量。

访问单独的位：

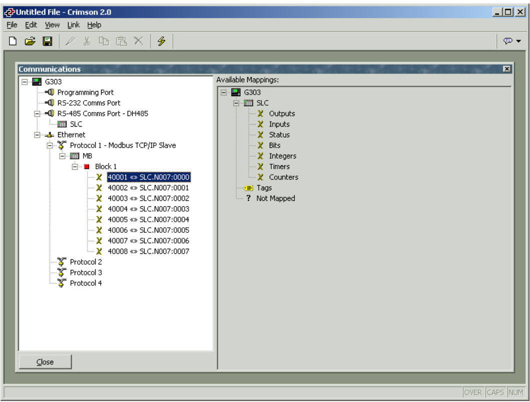


如果需要，你可以将数据块中的单独条目展开至组成位，并将不同的数据项目映射到每一个位。首先用右键点击所需的项，在弹出菜单中选择 Expand 指令，左手栏会被更新，将显示组成寄存器的每一个位，用同样的拖放方法完成数据映射。

协议转换：

除通过从站协议开放内部数据外，网关数据块也可以开放从其他远端设备获取的数据，或者是在两个主站设备间交换数据，这个独特的协议转换功能可使你使用简单低廉的设备来完成更加复杂的系统集成。

主站和从站：

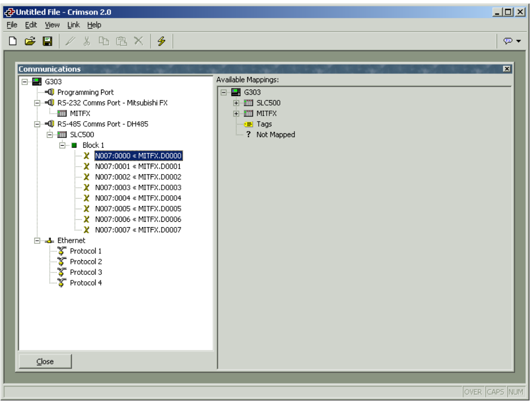


通过从站协议开放其他设备的数据，只是上面映射过程的简单延伸，只是它不是从右手栏中数据标签中拖动数据，你需要展开相应的主站设备，拖动所需的寄存器图标，你会被提示输入起始地址和需要映射的寄存器数量，映射就已入图创建完毕。

在这个例子中，Allen-Bradley 控制器中的寄存器 N7:00 至 N7:7 将通过 Modbus TCP/IP 中寄存器 40001 至 40008 开放以供访问。Crimson 会在 Modbus 读命令请求下自动读取 Allen-bradley PLC 内的数据，并将写入 Modbus 寄存器的数据，自动转换为写入 PLC 的数据。这个方式可以将简单的 PLC 连接到以太网。

主站和主站：

要完成两个主站设备间的数据交换，只需简单的选择其中一个设备，给该设备创建一个网关数据块，然后就可以映射其它设备中寄存器。Crimson2 将自动的按需要读写数据，并交换两个设备中的数据。下面的例子展示如何在 Mitsubishi FX 和 Allen-Bradley PLC 间转换数据：



哪种方法是对的呢？

一个经常遇到的问题是：我们应该在哪个设备中创建网关数据块？是在 Allen-Bradley 中？还是在 Mitsubishi 中？首先要明确的是，当处理同一个单一方向的传输时无须创建两个数据块，如果你在 Allen-Bradley 中创建一个数据块来读 Mitsubishi 中的数据，又在 Mitsubishi 中创建数据块读 Allen-Bradley 中的数据，那么你等于做了两次传输，这会降低整体速度。其次在哪一个设备中创建数据块都是可以的，通常应保持数据库中的数据块数量最少为好，如 Allen-Bradley 中的寄存器是一个连续范围，而 Mitsubishi 中的寄存器分散在整个地址，那么数据块就应创建在 Allen-Bradley 中，这样就可以减少数据块的数量。

数据转换：

你还可以利用网关数据块进行数学运算，而你的 PLC 就不必再处理了。例如：你可以读取 PLC 中的一个寄存器值，标定它，取平方根，再写回到另外一个 PLC 寄存器。要完成这个功能，请参考数据标签的章节，创建一个已映射的变量代表从设备读入输入值，再创建一个公式型变量代表输出值，写出数学表达式来执行运算。然后创建一个包含目标寄存器的数据块，最后将相应的公式型变量映射至目标寄存器，这就将计算值传回 PLC 了。

EDICT 用户须知：

Edict97 的用户一定会对他们所熟知的通讯模块的设置感到不解，答案是很简单的：Crimson 会自动的管理通讯模块，只有当系统需要时才会对数据进行读写。这就意味着：如果一个寄存器只是在一个特定页面被访问，那么只有当此页面被选择时，这个寄存器才会被读取。整个通讯的过程会被自动地优化。

在通讯方面 Crimson 和 Edict-97 还有不同之处：

- 从站协议不再在通讯模块中处理，而是在网关数据块中以映射数据项的形式替代。这就是说，同一个数据项可以在多个从站协议中开放，而不必进一步设置。
- 在 Crimson 中写操作是基于操作，而不是基于数值，这就是说，如果你向一个寄存器写 1，然后再写 0，保证将执行两次写操作。这样一来就避免了需要脉冲块等复杂的设置。
- 在 Edict-97 中，当需要将数据从一个设备传送到另一个设备，或者将设备转换后传回源设备，必须使用通讯刷新事件完成，而 Crimson 只需在网关数据块中简单设置即可。
- Crimson 的通讯处理结构比 Edict-97 具有更高的性能，它使用多任务操作系统和更大的缓冲区以保证最小的刷新时间。

设置数据标签

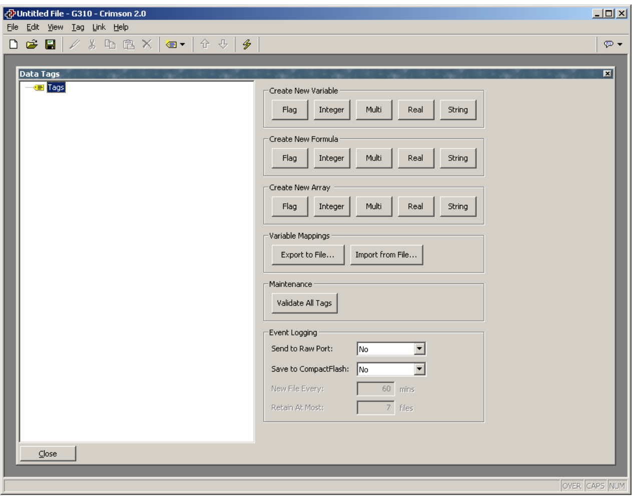
当你设置完成数据库中的通讯选项后，下一步就是要定义显示和操作的数据项。这些都是在主屏幕的数据标签图标中完成的。

关于数据标签：

数据标签是代表人机界面中数据的可命名实体。数据标签可以被映射为远端设备中的寄存器，这时 Crimson 将自动地读取相应寄存器的值，并用于相应的显示和操作。同样，如果你更改已映射的数据标签的值，Crimson 也将自动地将新值写入远端设备。

数据标签的种类：

当你打开数据标签窗口，在右手的窗中你将会看到三排按钮，它们是用来创建不同种类数据标签。开始时，可能会对按钮的数量有点儿吃惊，总共 15 个不同种类的标签，它们被分为 3 类，每类包括 5 种。



数据标签种类：

下面列出了 3 类数据标签的说明：

- 变量类：它代表屏内的单个数据项目。变量类可以映射至 PLC 的寄存器；可以设置为保持型，这样当人机屏断电后，可依然保持其数值。对变量类标签的属性设定，只包含一项内容，就是读/写特性。然而即使你在理论上可以对它写入，实际上该变量是只读的。（如果对此有些疑惑，读下去，你可以对比公式型标签，它没有这个属性。）
- 公式类：它代表可导出值。它是多个其它数据经过数学运算的结果。例如，一个公式型标签可代表 2 个油箱液位的高度和。一个公式型标签可以定义等于一个 PLC 寄存器的值，但它不是真正的映射，因为它不能被写入。为什么要这样限制？打个比方，一个公式型标签等于 Tank1+Tank2，那么如何向这个公式写值呢？

- 数组类：它代表人机屏中一些数据项目的集合。它们不能被映射至 PLC 的寄存器，它们可以代表屏中存储器中的一个数据表。它们可以用来存储配方数据；或统计分析数据的集合。它们不是用于简单应用，而是在复杂项目中的一个强大工具。

数据标签的类型：

每一类标签都有 5 种数据类型，每一种分别存储不同类型的数据：

- 标志型：它代表一个真/伪状态。当它被映射到远端设备的寄存器时，它大多代表内部线圈或数字 I/O 点。公式类标志型标签表示这类数据的逻辑运算和比较结果。
- 整数型：它代表带符号 32 位整数。它可以存储的数值在-2,147,483,648 和+2,147,483,648 间。即使被映射到的 PLC 寄存器是 16 位的，Crimson 在内部以 32 位方式进行处理，以确保可以处理更大的过程值。
- 多重状态型：它以整数的形式表示一些独立的状态。那么整数型可表示油箱的液位，而多重状态型，例如，可表示机器三种状态中的一种，停机，运行，暂停。区别是：多重状态型显示多个字符串中的一种，而整数型显示数字。
- 实数型：它代表 32 位单精度浮点数。它存储值的范围是：$\pm 10^{-38}$ 至 $\pm 10^{+38}$，精度是 7 个有效位。我们较少用实数型标签代表物理量（一般物理量的范围较窄），而多用于统计或其它数学运算。
- 字符串型：它代表包含一定数量字符的文本。它可用来存储配方名，或处理从原始串口接收的数据。它不能映射到 PLC 的寄存器，只能存储内部数据。

为什么需要使用数据标签？

给出了这么多的选项，你可能要问，为什么我要使用数据标签？毕竟 Crimson 允许直接将 PLC 的寄存器放置在显示页上，你也无需打开数据标签的图标，就可以编制一个简单的数据库，原因是：

- 标签可以使你对数据项命名，你可以清楚地知道你调用了 PLC 中那些数据。而且，PLC 中的数据有变动，或你转到一个不同的 PLC，那么你只需重新映射数据标签，而不必更改数据库中的所有内容。

- 标签可以避免重复输入相同的信息。当你创建了标签，你可以设定它如何在屏幕上显示，如一个整数型标签，Crimson 可以定义小数点位置和单位，以及附加的内容。当你在页面上放置一个标签，Crimson 便已知道以什么格式显示，无须更多设置。同样，如果你需要更改一些格式或单位，只需要在标签中更改，而无需在每个页面中逐个更改。
- 标签是从站协议中数据传输的关键。Crimson 对从站协议的处理方式是在人机屏内部开放相应数据项，这样就可以通过不同端口访问相同数据，例如，一个机器的设置可以通过内部的 SCADA 包来完成，也同时可通过以太网在远程站的相同数据包来完成。如果不用标签，整个数据交换将无法完成。
- 标签还用来完成 Crimson 内部的许多高级功能。如：告警，触发，数据记录和 Web Server 等功能。通过标签定义数据格式是使用这些功能的前提，是必须的。

总而言之，标签可在编程中自动完成许多任务，且节省时间。即使你不准备使用标签，随后的许多章节都涉及到它的内容，所以应完整地阅读这一节。

创建标签：

有两种方法可以创建数据标签，一种是选择标签图标后在数据标签窗口的右手窗中直接点击相应的按钮；另一种是使用工具栏中新标签按钮。无论用那一种方法，一个新的标签会加在标签列中。要重新命名标签，在左手窗中选择标签，键入新的名字。

标签的命名有一下规则：

- 标签不能包含空格和符号。
- 标签必须以字母或下划线开始。
- 随后的字符必须是数字，字母或下划线。
- 名字的长度不能超过 24 的字符。

编辑标签：

当在左手窗中选定了一个标签，右手窗会相应显示一些表，每个表都包含标签的一些属性。不同类型的标签具有不同的表，每个表的栏目也不同。

无论选择那一种标签，右手栏中的第一张表通常是数据表。这张表指示了标签代表什么数据，数据是怎样存储，数据是向远端设备传送还是从远端读取。表具体的内容是按标签的类型的不同而不同。

第二张表通常是格式表，它规定标签中的数据是以什么样的格式显示在人机界面的页面上，或其他任何媒介，如网页等。同一类型的标签具有一样的格式表形式，如所有整数型标签的属性是一样的。

其余的两张表是用来定义报警和触发的。但对于字符串型标签和数组型标签是没有这两项。报警表是用来定义哪些条件发生时，需要引起操作员的注意，或记录此事件。触发表的功能与报警表类似，不同的是，它不是记录事件，而是执行特定功能。

编辑属性：

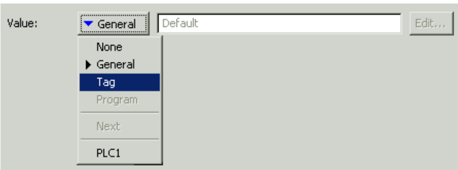
对于使用过 Windows 操作系统的人来说，大多数属性的编辑是非常容易地，有的属性需要输入数值；有的属性需要从下拉菜单中选择其中一项；有些标签类型的属性，由于有比较复杂的选项，下面将作一个介绍。

表达式属性：

表达式属性可以被设为：

- 一个常数。
- 一个数据标签的值。
- 远端通讯设备中寄存器的值。
- 以上数据经数学运算的表达式。

在缺省设置下，在属性标志后面的一个带箭头的按钮中，显示目前栏目是在通用模式下，在按钮右边的编辑盒中，变灰显示的字符串指示目前定义的属性。



如果你熟悉 Crimson 中表达式的句法（这将在书写表达式章节中详述），你可用直接键入表达式的方式来编辑属性。对于大多数用户，可以选择点击按钮，从弹出的菜单选项中选取：

- 选择 Tag，会弹出一个包含数据标签列表的对话盒。你可以选择其中的标签来控制这个属性。有时，你可以直接在这里创建一个新标签，并定义它的基本属性。但如果所编辑的属性直接属于其他数据标签，则不能创建。如果不这样，你会太容易忘记你在编辑哪一个标签！

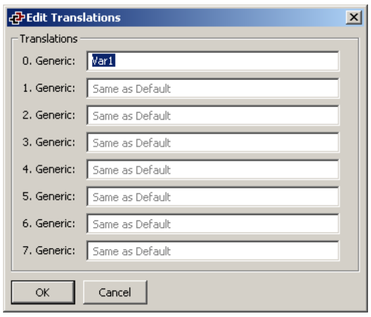
- 选择一个设备名，将会弹出一个对话框，在对话框中你可以选择远端通讯设备中的寄存器。不同的通讯设备按创建的顺序在菜单底部排列。
- 选择 Next，将置标签等于你上一次所选寄存器的下一个地址。例如：如果你上一次设置标签的属性为 PLC1 的 N7：10，当你选择 Next，当前标签的属性被设为 PLC1 的 N7：11。

可译字符串：

Crimson 的数据库支持多语种操作，因此所有在人机界面上显示的字符串都可以按不同的语言自动显示。要定义这些字符的翻译，在包含字符串内容的编辑盒的右手边，有一个标记为 Translate 的按键，



点击此按键就进入了翻译对话框...



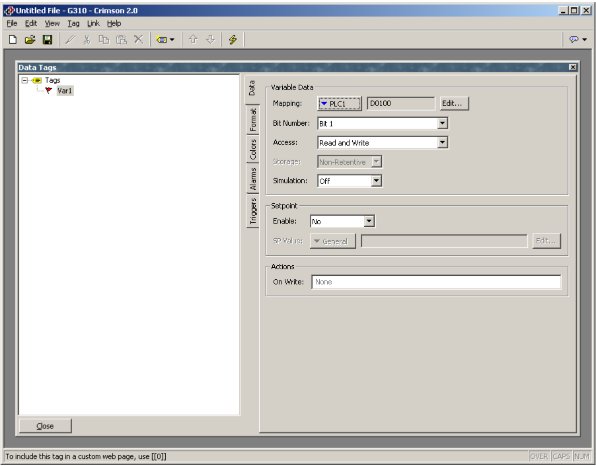
如果你没有输入相应语言的文字，而语言又由操作员在以后选择，Crimson 将使用美国英语作为缺省设置。关于如何使用一个按键或菜单来选择不同语言，请参考用户界面章节。

编辑标志型标签：

你可以回想起标志型标签是代表真/伪值。下面的部分，将描述显示在数据标签窗口的右手的各种表格，它们用来定义各种不同类型的标志型标签。

数据表（变量类）

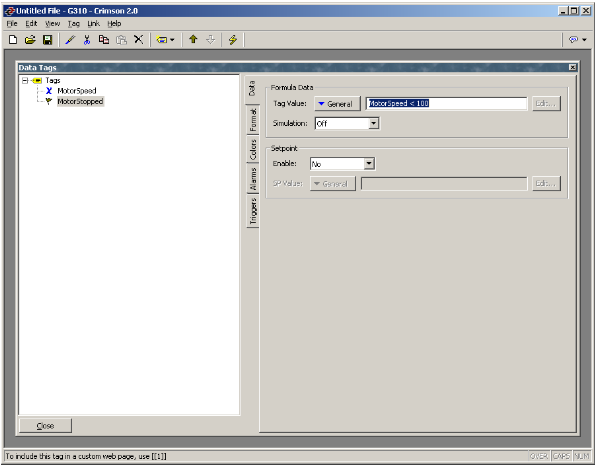
标志型变量的数据表包含以下属性：



- 映射属性：它是用来设定将标签映射至远端设备中的寄存器，或仅供设备内部使用。如果你点击带箭头的按键，并从弹出菜单中选择一个设备名，你将会看到一个包含可选 PLC 寄存器的对话框。
- 位数属性：它是用来将标志型变量映射至 PLC 寄存器的某一位。它也是用来指示数据标签将访问寄存器中的哪一位。
- 访问属性：它是用来设定用哪一种传输方式来传递已映射的变量。你可以规定数据可同时被读和写，或者规定数据是只读或只写。只写型标签可以避免不必要的读操作，这些标签通常被设置为可保持，因为这些数据是不能从 PLC 中获取，因此必须存储在人机界面中。
- 存储属性：它是用来设置是否使用屏内部的 FLASH 存储器来保存数据标签值，这样在掉电情况下可以始终保持数据。
- 设定点属性：它是用来设置该标签是否有设定点，以及设定点是多少。设定点可被用在不同的报警模式中，可将实际的标签状态与设定状态相比较。例如：一个标签代表一个检测电机转速的速度开关的状态，这样我们可以设置一个报警来指示电机起动失败。

数据表（公式类）

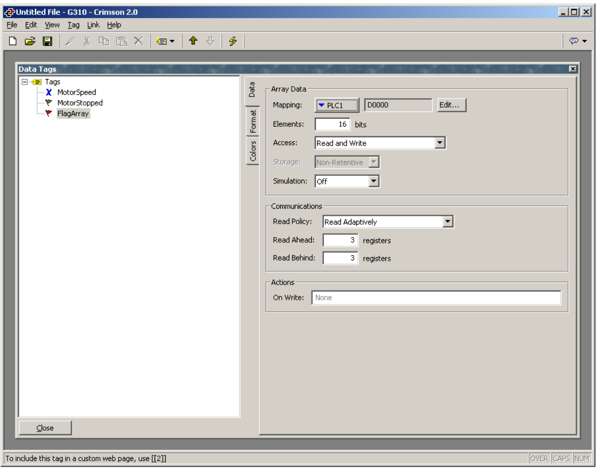
一个公式类标志型数据变量包含以下属性：



- 标签值属性：它用来设置该标签所代表的值，通常它是其它标签或 PLC 寄存器值的逻辑运算组合，或是数值的比较结果。在上面的例子中，当电机转速超过设定值，该标签为真。
- 设定点属性：它的用法与变量类标志型标签相同。

数据表（数组类）

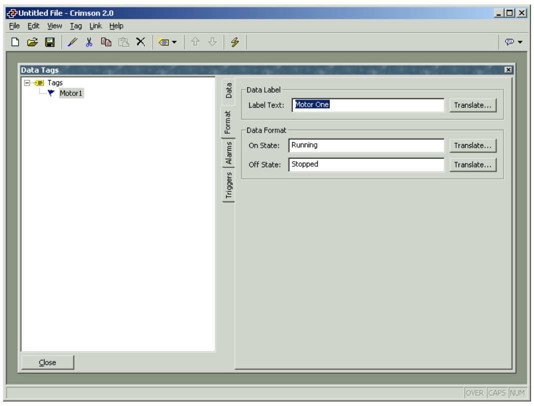
一个数组类标志型数据变量包含以下属性：



- 单元属性：它用来设置该数组包含多少个数据项。数组单元是用方括号来索引的，如：Array[0]是指数组的第一个数据单元，Array[n-1]是指最后一个数据项，n 就是该属性的设置值。
- 存储属性：它的用法与变量类标志型标签相同。

格式表：

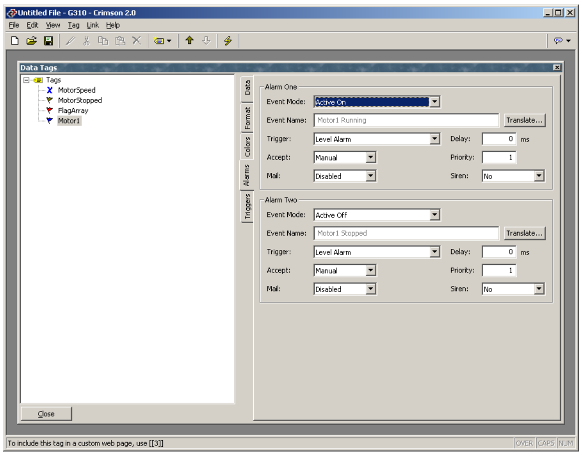
标志型标签的格式表包含以下属性：



- 标注文本属性：它所规定的文本内容，将在页面中显示在标签值的前面。标注与标签名是不同的，前者可以在转译成多种语言，而后者是不变的，且不能显示在屏幕上。
- ON 状态和 OFF 状态属性：它规定当标签值为零或非零值时，在屏幕上分别显示的文本内容。当你输入了在 ON 状态下显示的文本，Crimson 会自动按反义或以往创建记录来产生 OFF 状态下的文本。

报警表：

标志型标签的报警表包含以下属性：



- 事件模式属性：它用来规定报警触发的逻辑模式。该项属性有以下可用模式：

模式	**报警触发状态**
Active On	标签值为真
Active Off	标签值为伪

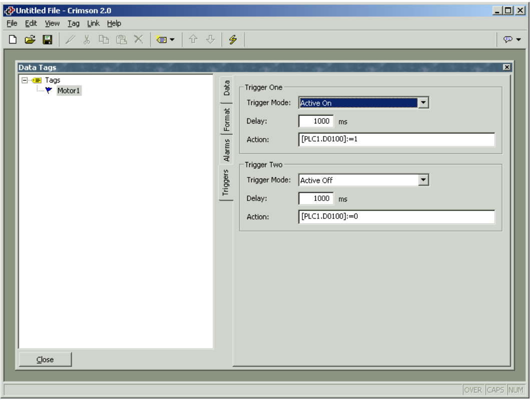
当定义了设定点后，该属性可以定义为以下模式：

模式	**报警触发状态**
Not Equal to SP	标签值不等于设定点
Off when SP On	在设定点 ON 状态下标签不响应
On when SP Off	在设定点 OFF 状态下标签不响应
Equal to SP	标签值等于设定点

- 事件名称属性：它是用来定义显示在报警窗口或事件记录器中的报警名称。当事件模式选定后，Crimson 设置一个基于标签标注名称的缺省名称。
- 触发属性：它是用来设置报警是边界沿触发，还是水平触发用。在前一种状态，当所设置的条件第一次成立时，报警将被触发。在后一种状态，只要所设置的条件成立，将持续报警。这个属性也可以定义为事件模式，在此状态下，报警是由边界沿触发，但并不产生一个报警，而是将该事件记录在 G3 的内部存储器。
- 延时属性：它是用来设定在报警触发以前，报警条件必须保持多长时间。在边沿触发和事件模式下，此属性代表报警条件成立的次数，例如：一个报警被设为指示电机在接到起动命令后并没有运转，此项属性可用来设定报警触发前电机可以起动的次数。
- 复位属性：它是用来设定用户是否需要对一个报警进行手动复位，报警显示随之清除。边沿触发模式通常必须用手动复位。
- 优先级属性：它是用来控制在报警窗口中各报警显示的顺序。数字越小的报警，越靠近窗口的顶部。
- 打印属性：该属性待以后扩展。
- 警笛属性：它是用来规定当该报警产生后，G3 内部的蜂鸣器是否鸣响。当蜂鸣器鸣响时，屏幕的显示也会闪烁，以更好地提醒操作员注意。

触发表：

标志型标签的触发表包含以下属性：



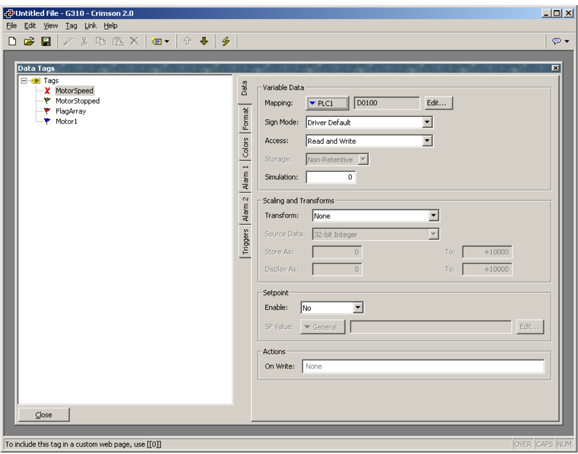
- 触发模式属性的功能与报警表中的描述相同。
- 延时属性的功能与报警表中的描述相同。
- 动作属性：它是用来设置当触发发生后执行什么动作。请参考书写动作章节中关于句法的描述，以及 Crimson 所支持的动作。

编辑整数型标签：

前面我们介绍过整数型标签代表含符号的 32 位数值。下面的章节将描述当我们编辑某类整数型标签时，显示在数据标签窗口的右手边的各种表。

数据表（变量类）

整数型变量的数据表包含以下属性：



- 映射属性：它是用来设定将标签映射至远端设备中的寄存器，或仅供设备内部使用。如果你点击带箭头的按键，并从弹出菜单中选择一个设备名，你将会看到一个包含可选 PLC 寄存器的对话框。

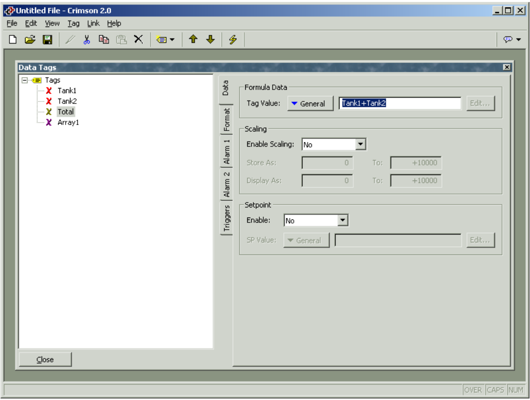
- 符号模式属性：当从远端设备中读取 16 位数据时，它是用来更改通讯协议中的缺省设置的。通常通讯协议会根据数据在设备中的使用状况来确定是按带符号或不带符号来处理数据。当你需要更改时，设置该属性即可。
- 访问属性的功能与标志型变量的描述相同。
- 存储属性的功能与标志型变量的描述相同。
- 标定和转换属性是当从远端设备中读/写数据时，用来进行数据变换的。当选择线性标定模式时，可以按 PLC 中变量的上，下限值存储数据；而显示时按指定的相应范围来显示在屏幕上。其它模式说明如下：

模式	**说明**
BCD to Binary	将 BCD 码转换为二进制码
Binary to BCD	将二进制码码转换为 BCD
Swap Bytes in Word	16 位数据中高/低字节互换
Swap Bytes in Long	32 位数据中高/低字节互换
Swap Words	32 位数据中高/低字互换
Reverse Bits in Byte	第 0 位到第 7 位的值取反
Reverse Bits in Word	第 0 位到第 15 位的值取反
Reverse Bits in Long	第 0 位到第 31 位的值取反
Invert Bits in Byte	第 0 位到第 7 位的值颠倒
Invert Bits in Word	第 0 位到第 15 位的值颠倒
Invert Bits in Long	第 0 位到第 31 位的值颠倒

- 设定点属性：它是用来设置该标签是否有设定点，以及设定点是多少。设定点可被用在不同的报警模式，可将实际的标签状态与设定状态相比较。例如：用一个标签代表某容器的温度，并设置一个设定点来指示所需的温度，当容器的温度偏离目标值一定距离时，我们就可以触发一个报警。

数据表（公式类）

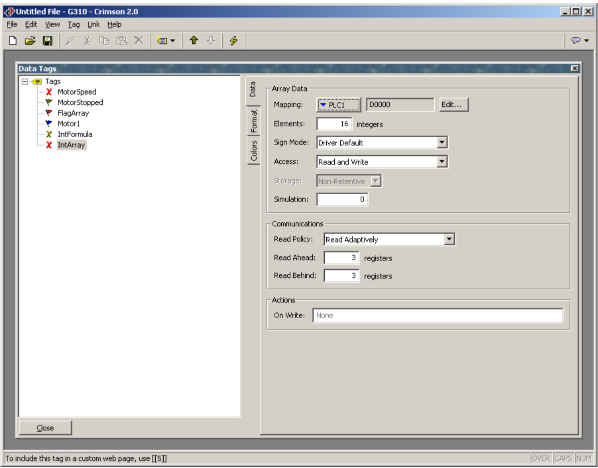
公式类整数型标签的数据表包含以下属性：



- 标签值属性：它是用来设置标签所代表的值。通常它是由各种它是其他数据标签与各种数学运算的组合。上面的例子中，标签设置为两个油箱液位的高度和，所以可以指示总体储油量。
- 标定和转换属性：它与整数型变量的描述相同。
- 设定点属性：它与整数型变量的描述相同。

数据表（数组型）

数组类整数型标签的数据表包含以下属性：

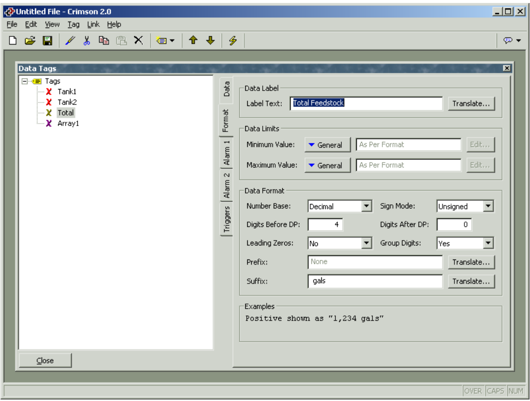


- 映射属性：它是设置是否将变量标签映射至远端设备中的寄存器，或供内部人机界面使用。如果你点击带箭头的按键，并从弹出菜单中选择一个设备名，你将会看到一个包含可选 PLC 寄存器的对话框。
- 单元属性：它是设置在数组中包含多少个数据项。数组单元是用方括号来索引的。如：Array[0]是指数组的第一个数据单元，Array[n-1]是指最后一个数据项，n 就是该属性的设置值。

- 符号模式属性：当从远端设备中读取 16 位数据时，它是用来更改通讯协议中的缺省设置的。通常通讯协议会根据数据在设备中的使用状况来确定是按带符号或不带符号来处理数据。当你需要更改时，设置该属性即可。
- 访问属性的功能与标志型变量的描述相同。
- 存储属性的功能与标志型变量的描述相同。
- 通讯读取规定属性：它是用来指示将读取寄存器中的哪一位并且输入到数组中。选择读取整个数组，将读取整个寄存器并输入到数组中；选择人工设置读取，将读取寄存器指定位并输入到数组中；选择相关读取，将读取寄存器的指定位以及相连的数据位，输入到数组中。

格式表：

整数型标签的格式表包含以下属性：

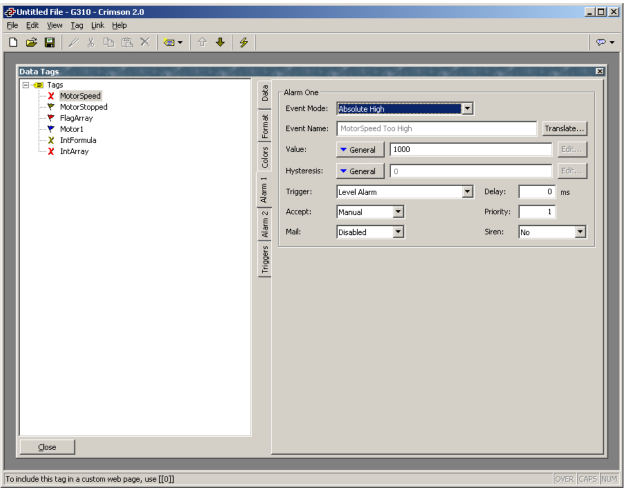


- 标注文本属性：它所规定的文本内容，将在页面中显示在标签值的前面。标注与标签名是不同的，前者可以在转译成多种语言，而后者是不变的，且不能显示在屏幕上。
- 最小值和最大值属性：它是用来定义数据输入的有效范围，并为与数据标签相连的各种图形元件提供类似的数据界限。例如：当在一个柱状标尺元件中标定一个标签的显示值。
- 数学进制属性：它是用来规定标签是以什么数学进制显示在屏幕上，十进制，十六进制，二进指，八进制或密码。十进指数可以是带符号或不带符号，而其它进制数则都以无符号数运算。密码从安全的方面考虑将以星号显示。

- 符号模式属性：它是用来设定数据标签值是否要加正/负符号。如选择硬符号选项，将会在数值前加入正号或负号；如选择软符号选项，正号将不显示，以空格替代。
- 小数点前位数属性：它设置小数点前有多少位。如果是无小数位，它表示总位数。
- 小数点后位数属性：它设置小数点后有多少位。如果不是十进制数，此项将不支持。
- 起始零位属性：它设置是否显示数值的起始零位，或以空格替代。
- 数字位组属性：它设置是否将数据按三位一组的形式以逗号分隔，对于其它进制的数值，也按同样形式分隔。
- 前缀属性：它是用来设置显示在数值前的字符。通常它是设置数据单位。
- 后缀属性：它是用来设置显示在数值后的字符。通常它也是设置数据单位。

报警表：

整数型和公式型变量的报警表包含以下属性：



- 事件模式属性：它用来规定报警触发的逻辑模式。该项属性有以下可用模式：

模式	**触发条件**
Data Match	标签值等于报警值
Data Mismatch	标签值不等于报警值
Absolute High	标签值大于报警值
Absolute Low	标签值小于报警值

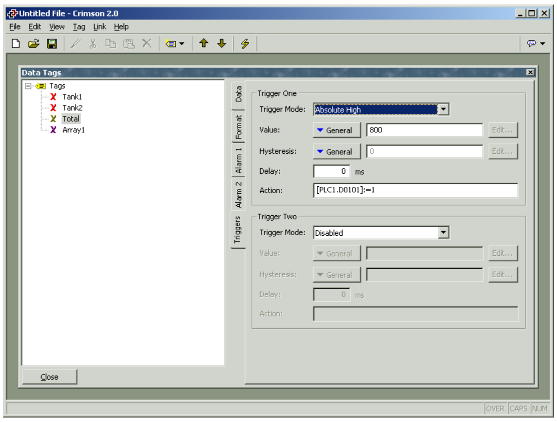
下列模式仅适用于设定了设定点后：

模式	**触发条件**
Deviation High	标签值超出设定点值加报警值
Deviation Low	标签值超出设定点值减报警值
Out of Band	标签值超出设定值加报警与减报警值之间的范围
In Band	标签值在设定值加报警与减报警值之间的范围

- 数值属性：它设置报警触发的绝对值或相对于设定点的偏差。它决定于事件模式的设置。
- 回滞属性：它是用来防止在接近报警值时发生震荡状态。例如：在绝对高报警状态下，当标签值高于报警值时将触发报警，但只有在标签值低于报警值减回滞值时，报警才复位。这个属性是保持报警状态，而不是更改报警点。

触发属性：

整数型和公式型变量的触发表包含以下属性：



- 触发模式属性与报警表中的描述相同。
- 延时属性与标志型标签的报警表的描述相同。

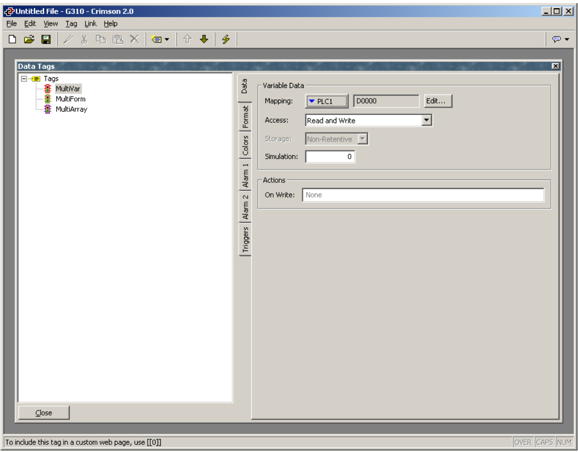
- 动作属性：它是用来设置当触发发生时，执行什么动作。可参考在编写动作章节中关于 Crimson 所支持的各种动作的句法说明。

编辑多重状态变量：

多重状态标签是代表一个 32 位的带符号值，但它是用来选择一个至多个字符串文本。下面的章节将描述在编辑多重状态标签时，显示在数据标签右手边的各种表格。

数据表（变量型）

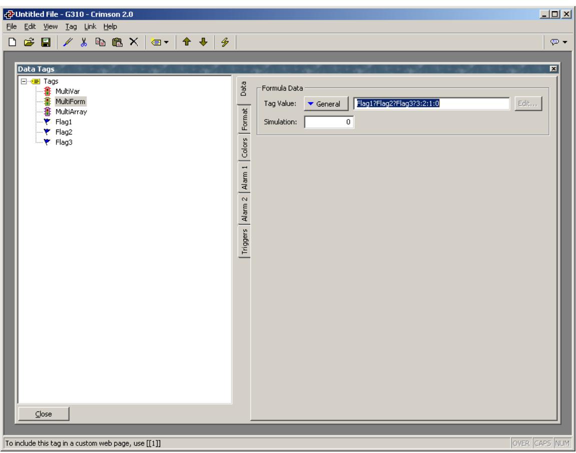
多重状态变量的数据表包含以下属性：



- 映射属性：它是设置是否将变量标签映射至远端设备中的寄存器，或供内部人机界面使用。如果你点击带箭头的按键，并从弹出菜单中选择一个设备名，你将会看到一个包含可选 PLC 寄存器的对话框。
- 访问属性与标志型变量的描述相同。
- 存储属性与标志型变量的描述相同。

数据表（公式型）

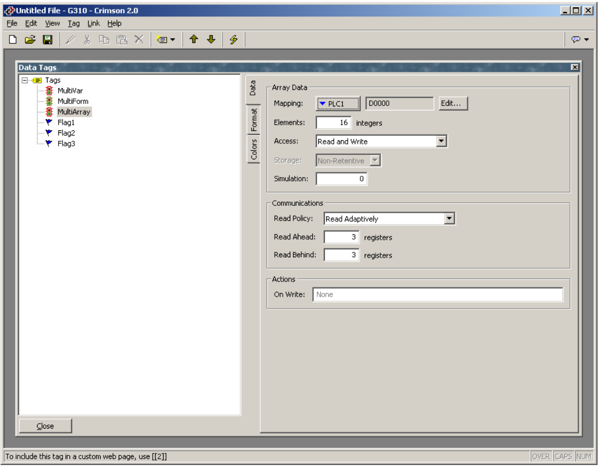
公式类多重状态型变量的数据表包含以下属性：



- 标签值属性：它是用来设置标签所代表的值。通常它是由各种它是其他数据标签经各种数学运算的组合。上面的例子中，数据标签可以等于 1，2 或 3，它决定于三个不同标志型变量的状态。关于操作符?: 的说明，可参考编写表达式章节。

数据表（数组）

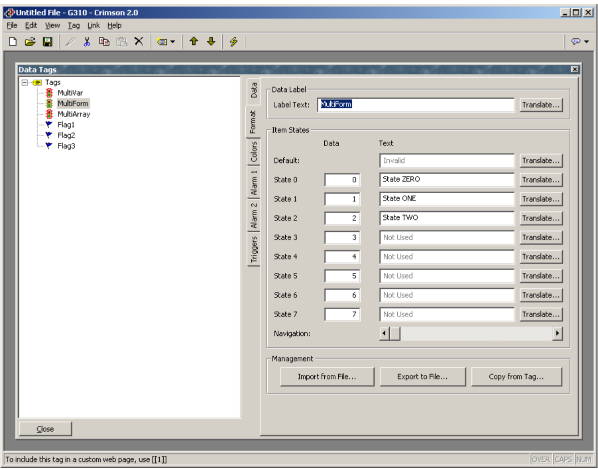
数组类多重状态型变量的数据表包含以下属性：



所有此类属性与数组类标志性变量相同。

格式表：

多重状态型变量的格式表包含以下属性：



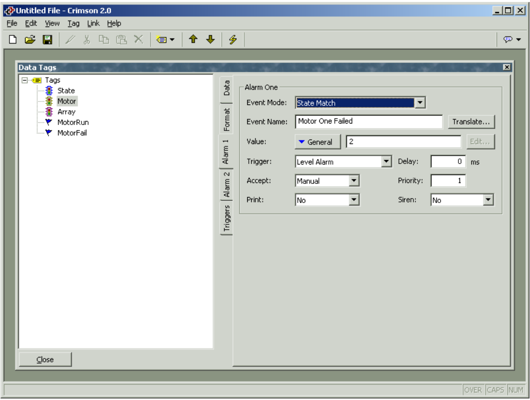
- 标注文本属性：它所规定的文本内容，将在页面中显示在标签值的前面。标注与标签名是不同的，前者可以在转译成多种语言，而后者是不变的，且不能显示在屏幕上。

- 项目状态属性：它是用来定义 8 个数值所代表的标签的不同状态。每一个状态都与一个整数值相关联，并决定显示什么样的字符串。最少要定义两种状态，其余的，如不需要，可保持在缺省状态。

- 缺省属性：它是用来定义当标签值超出所定义的范围时，在屏幕上所显示的文本。

报警表：

变量类和公式类多重状态型标签的报警表包含以下属性：



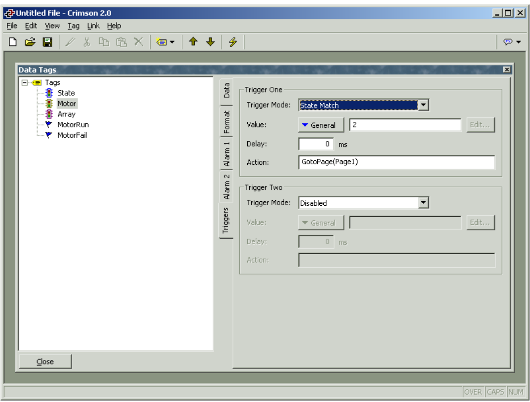
- 事件模式属性：它用来规定报警触发的逻辑模式。该项属性有以下可用模式：

模式	触发状态
State Match	标签值等于报警值
State Mismatch	标签值不等于报警值

- 数据值属性：它是用来设置报警中的比较值。
- 其它属性与标志型标签的报警表描述相同。

触发表：

变量类和公式类多重状态型标签的触发表包含以下属性：



- 触发模式属性与报警表中的描述相同。
- 延时属性与标志型变量的报警表的描述相同。

- 动作属性：它是用来设置当触发发生时，执行什么动作。可参考在编写动作章节中关于 Crimson 所支持的各种动作的句法说明。

编辑实数型标签：

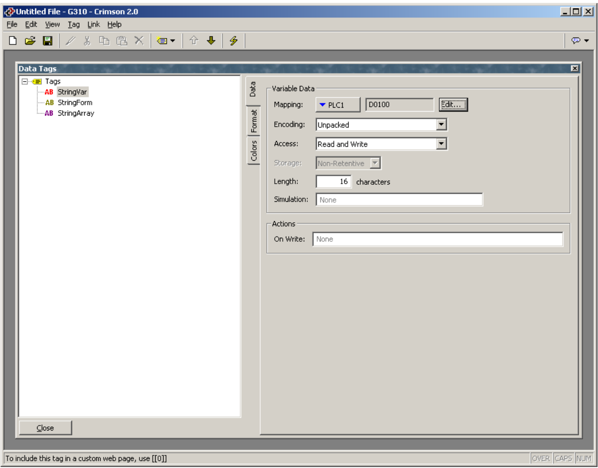
实数型标签代表单精度浮点数。实数型标签的所有设置表与整数型标签完全相同，仅仅是在报警和触发表的数值属性和回滞属性中是包含小数。你可参考整数型标签的章节。你会注意到整数型标签的有些选项在实数型标签中是没有的。

编辑字符串型标签：

字符串型标签代表文本，它是由一些单独字符组成。下面的章节将描述当编辑字符串型标签时，显示在数据标签右手边的各种表格。

数据表（变量类）

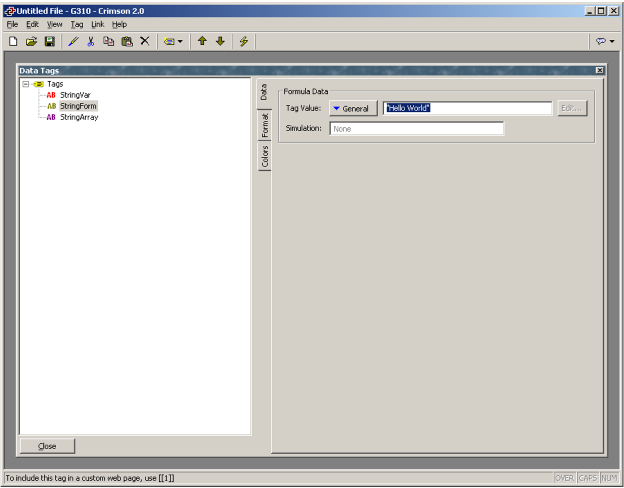
字符串型变量的数据表包含以下属性：



- 存储属性与标志型变量中的描述相同。
- 长度属性：它是用来定义在此字符串中需要存储多少个字符。如果你需要对此变量保持存储，就必须设定数值。如果字符串仅保持在人机屏的 RAM 中，而不必写入闪存，则不需限制长度。

数据表（公式类）

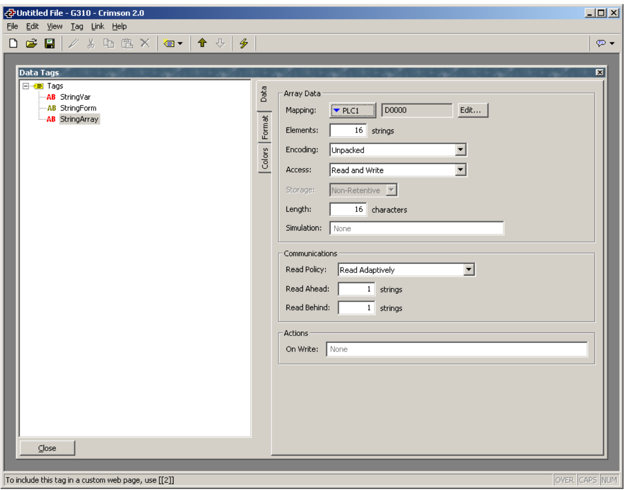
公式类字符串型变量的数据表包含以下属性：



- 标签值属性：它是用来设置标签所代表的值。通常它是由各种其他数据标签经各种数学运算的组合。上面的例子中，标签值等于两个字符串变量之和，中间间隔一个空格。关于字符串中所用的操作符和功能，可参考编写表达式和函数索引章节。

数据表（数组类）：

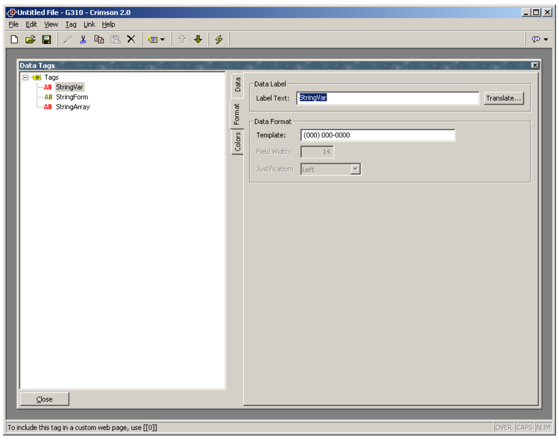
数组类字符串型变量的数据表包含以下属性：



- 单元属性和存储属性与数组类标志型变量中的描述相同。
- 长度属性与字符串变量中的描述相同。

格式表：

字符串型变量的格式表包含以下属性：



- 标注文本属性：它所规定的文本内容，将在页面中显示在标签值的前面。标注与标签名是不同的，前者可以在转译成多种语言，而后者是不变的，且不能显示在屏幕上。
- 模板属性：它是规定一种字符串的基准模板，可指定在每个位置上可出现什么字符。模板规定后，数据的输入将受到字符种类的限制。下面的表格将显示模板中可包含的特定字符的含义：

模板中的字符	允许字符				
	A-Z	**a-z**	**0-9**	**Space**	**Misc**
A	可	—	—	—	—
a	可	可	—	—	—
S	可	—	—	可	—
s	可	可	—	可	—
N	可	—	可	—	—
n	可	可	可	—	—
M	可	—	可	可	—
m	可	可	可	可	—
O	—	—	可	—	—
X	可	可	可	可	可

在“Misc”列中引用的额外字符有：

， . ; + - = ! ? % / $

在表格没有包括的字符会在显示中被逐字复制。

例如：要输入一个美国电话号码，可使用一个模板：

（000） 000-000

括号，空格和划线将都被显示，而由“0”指示的十个数字都会存放在字符串中。同样，如果使用此模板的数据是可输入的，光标在左右移动时将跳过非数字位置，并且只能输入数字字符。

- 长度属性：在使用模板的场合，它设定在显示字符串时多少字符将被保留。如果字符串变量设为保持型，那么它的功能与数据表中的长度属性相同，但不是必须的，你可以加入一些空格。

超过两个报警：

如果在你的应用中一个标签需要两个以上的报警或触发，你可定义一个公式型标签等于原始标签，并以别名的形式设置额外报警。例如：你有一个变量叫 Level1，映射到 PLC 中 N7：100。如你需要创建第三个报警，首先创建一个公式型变量，LevelAlias，设定它等于 Level，你就可以在这个标签中设定额外报警了。

Edict 用户须知

Edict97 的用户必须注意：

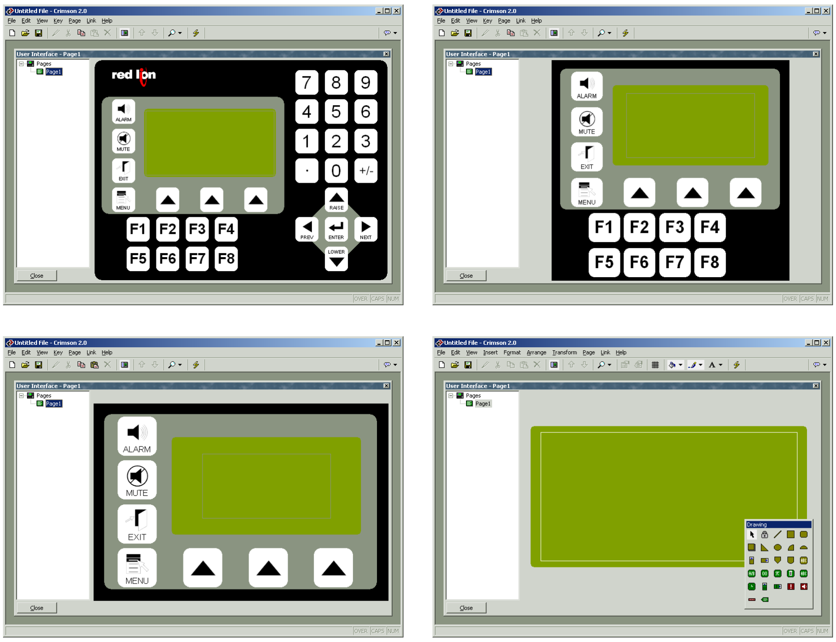
- Crimson 的数据标签窗口是用来执行 Edict97 中命名数据窗口，报警扫描和触发表中的所有功能。
- Crimson 没有将常数作为一个单独的标签类。Edict 使用常数所完成的优化，现在 Crimson 可以自动完成。常数可以用公式的方式完成。
- Crimson 将报警和触发与标签联系在一起，而不是将它们定义为一个表达式，如果需要监控一个表达式，可定义一个公式类标签等于表达式，再设定报警或触发。

设置用户界面:

现在你已经设置了通讯口,创建了你需要显示的数据标签,下面你需要创建显示页面,使用户可以浏览数据,并修改数据。页面的操作首先在主界面上选择用户界面图标.

显示控制:

按缺省设置，用户界面窗口将显示 G303 的整个前面板，包括显示和所有按键。如果你想要显示更多 G303 显示窗口，你可以使用下面四种缩放尺寸：



你可以看到，随缩放尺寸的增大，按键显示的越少，显示窗口越大。缩放尺寸的控制是在 View 菜单中，或使用工具栏中的放大镜图标，或 Alt 键加数字键 1~4。

其他外观选项：

除了缩放控制外，View 菜单中还包括以下选项：

- Page List 命令可以控制用户界面窗口中左手窗口的显示或隐藏。如果 Page List 命令关闭，则可以有更多空间来显示和编辑页面。F4 键可以转换页面列表的开关。
- Hold Aspect 命令是用来控制显示图形的外观比率。如果 Hold Aspect 命令打开，一个图形，如在 G303 上显示的圆，将与在电脑中的显示完全相同。而如果没有选择该模式，Crimson 会扩展显示页面以使用更多电脑屏幕，但会有一些变形。

页面编辑中的其它可用选项将在下面将介绍。

使用页面列表：

要创建、删除或重命名显示页面，首先点击用户界面窗口的左手窗，再从 Page 菜单中选用相应的命令即可。或者，你可以用鼠标右键点击所需页面，然后从弹出菜单中选取命令。

要选取一个页面，可以点击页面列表中的相应页面名，或使用工具栏中的上/下箭头键；再或者使用 Alt+左键和 Alt+右键的组合来上下移动所选的页面。所有这些键在任何窗口中都可使用。

显示编辑工具盒：

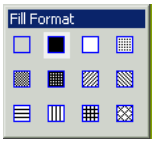
要编辑显示页面的内容，首先如上选定显示页面，然后点击代表 G3 页面的绿色方框，这时页面的四周出现一个白色方框，这代表页面已经选中，一些工具盒将会出现。

绘图工具盒：



绘图工具盒是用来在页面上添加不同的元素，我们称之为元件。前两个图标是用来控制插入模式，其它的图标代表不同的元件。所有显示黄色的元件是基本的几何和动态图形；显示绿色的元件是多功能元件，它可以调用数据标签中的格式和其它信息来控制其操作；显示红色的元件是系统元件，如动态报警浏览器。所有工具盒中包含的元件都可以从 Insert 菜单中选取。

填充格式工具盒：



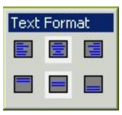
填充格式工具盒是用来控制显示元件的填充式样。如果选择一个或多个元件，然后点击选择一个填充式样，则所有选择的元件会显示同样的式样。如果当前没有选择元件，而点击一个式样，则以后新创建的元件都将显示此式样。不同的选项可以通过菜单中的 Format 来访问，或通过工具栏中的油漆桶图标访问。

线型格式工具盒：



线型格式工具盒是用来控制显示元件外轮廓线的颜色的。如果先选择一个或多个元件，然后点击一个线型颜色，则所选的元件都显示同样的颜色。如果没有选择元件，而点击线型颜色，则是设置新创建的元件线型颜色。不同的选项可以通过菜单中的 Format 来访问，或通过工具栏中的刷子图标访问。

文本格式工具盒：



文本格式工具盒是用来控制文本元件的水平或垂直对齐的。如果先选择一个或多个元件，然后点击一个对齐方式，则所选的元件都按所选方式对齐。如果没有选择元件，而点击一个对齐方式，则是设置新创建的元件对齐方式。

不同的选项可以通过菜单中的 Format 来访问。

前景工具盒：



前景工具盒是用来控制文本元件的前景颜色。如果先选择一个或多个元件，然后点击一个颜色，则所选的元件都按所选颜色显示。如果没有选择元件，而点击一个颜色，则是设置新创建的元件显示颜色。不同的选项可以通过菜单中的 Format 来访问。

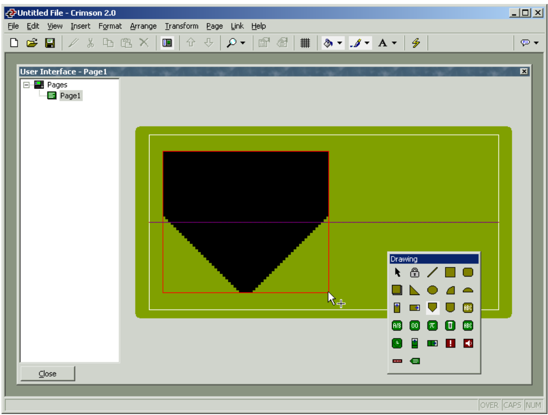
背景工具盒：



背景工具盒是用来控制文本元件的背景颜色。如果先选择一个或多个元件，然后点击一个颜色，则所选元件的背景都按所选颜色显示。如果没有选择元件，而点击一个颜色，则是设置新创建的元件显示颜色。不同的选项可以通过菜单中的 Format 来访问。

添加显示元件：

在页面中添加显示元件，可在绘图工具盒中点击所需的图标，或在 Insert 菜单中选择所需的选项，此时鼠标光标将变为带十字的箭头，在显示窗口中的适当位置拖动鼠标，元件就显示在页面上了。

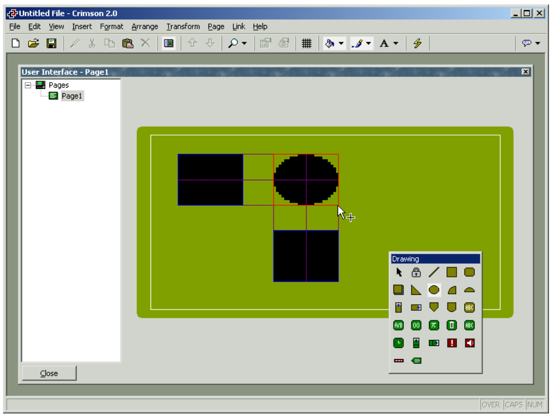


自动对齐：

如果你打开了 View 菜单中的 Smart Align 功能，Crimson 将显示引导线来帮助新元件和现存元件的对齐。在上面的例子中，水平的虚线指示油箱元件的中心与屏幕的垂直中心对齐。经过简单的练习，这个功能可以轻松的对齐所创建的元件，而不需要反复调整元件位置以保持对齐。

在下面的自动对齐例子中，新创建的椭圆与现有的两个长方型对齐。引导线代表图形的两个边线和中心线，显示边和中心都已对齐。

红色的矩形框指示新建的元件，而蓝色的矩形框指示引导线所对齐的元件。



当元件被移动或缩放时，自动对齐功能同样适用。

键盘选项：

在创建显示元件时，可使用以下键盘选项：

- 当按住 Shift 键，并在页面拖动放置元件，此时元件的中心将置于鼠标光标的起点，元件的一个端点将置于光标的终点。（如果不理解，可实际试一下，这样会更容易明白！）这在绘制对称图形时，更容易设置中心点。
- 当按住 Ctrl 键，并在页面拖动放置元件， 此时元件的水平和垂直尺寸将一致。此功能在绘制圆形和正方形非常有用。

这些功能在元件被缩放时同样适用。

锁定插入模式：

在绘图工具盒中的挂锁图标是用来添加多个同型的元件，而不需要重复选择图标。要取消锁定模式，再次点击挂锁图标，或按 Esc 键。完成同样的操作可通过 Insert 菜单中的 Lock Mode 命令完成。

选择元件：

要在页面中显示一个元件，只需移动鼠标指向目标元件，以左键点击即可。你可以注意到当鼠标箭头在元件上移动时，会出现一个蓝色的边框，以提示你所选元件。当实际选择后，方框变为红色，并出现一个手柄，你可以按需要缩放元件。如果你要选择的元件隐藏在其它元件后，按 Alt 键来选择。

要选择多个元件，可以用拖动方框的形式来选择；也可以用按住 Shift 键，同时用鼠标依次点击元件即可。选择完元件，元件周围出现一个红色的方框，手柄可以用来成组缩放元件。只要不小于元件的最小尺寸，元件相关的尺寸和位置将随意变化。

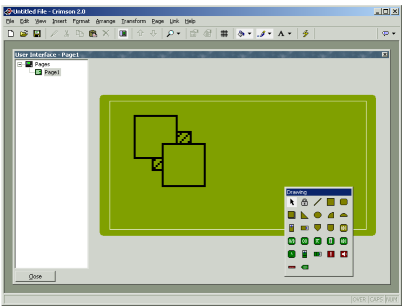
移动和缩放：

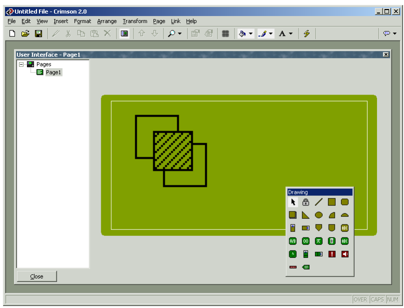
要移动元件，首先选择元件，然后拖动它们到页面指定位置。如果打开了自动对齐功能，将出现引导线来帮助你与其它元件对齐。在移动元件时，按住 Ctrl 键，会在原始位置留下一个元件副本，也就相当于复制元件。你可以使用方向键按一个像素点距离来移动元件，如果同时按住 Ctrl 键，将以 8 个像素点的距离来移动元件。

要缩放元件，首先选择元件，然后拖动相应的手柄到指定位置。同样，如果打开了自动对齐功能，将出现引导线来帮助你与其它元件对齐。Shift 和 Ctrl 键可用来改变拖动的效果，它与添加显示元件中的功能相同。注意：Crimson 将限制缩放的操作，以保证元件在屏幕容许的最大范围，和最小范围。

重排元件：

元件在页面上是按 Z 方向顺序放置的。它是由元件绘制的顺序来定义的，所以有些元件会显示在另一个元件的前面或后面。在下面的第一个例子中，带阴影的方块显示在其它两个实体方块的后面，所以它处于 Z 方向顺序的底层。在第二个例子中，它被移到顺序的上层，也就显示在其它图形的前面了。





要在 Z 方向移动图形，首先选择图形，然后使用 Arrange 菜单中的不同命令来完成。Move Forward 和 Move Backward 命令是将图形向前和向后移动一层。Move To Front 和 Move To Back 命令是将图形移动到最上层和最底层。同时，如果你的鼠标是带有滚轮的，向前滚动滚轮是将图形移向底层，向下滚动则相反。

编辑元件：

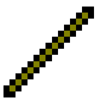
除了以上的方法外，元件还可以用以下方法来编辑：

- 在 Edit 菜单中的多种剪贴板命令（如：剪切，复制和粘贴），或是相应的工具栏图标，都可以在页面中或页面间复制和移动图形。Duplicate 命令是复制和粘贴命令的组合。注意：当执行粘贴命令时，Crimson 将新的图形与原图形间有一个偏移。
- 多种格式属性（如：填充式样，外框线颜色，文本对齐等）可以通过点击相应工具盒中的不同按键，或从 Format 菜单中选取相应命令完成。如果选择了多个元件，Crimson 将同时改变多个元件的属性。
- 编辑更详细的元件属性，可以双击元件，或使用 Edit 菜单中的 Properties 命令，将弹出一个对话框，你可以访问所有的元件属性。每个元件的相关属性将在下面的描述中说明。

元件描述：

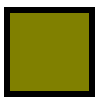
下面的章节将描述绘图工具盒中的每一个元件：

直线元件：

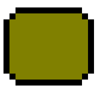


直线元件是两点间的一条直线。它唯一的属性是直线的线型。另外，实体的颜色也显示在直线工具盒中，在属性对话盒中可以选择几种不同虚线线型。

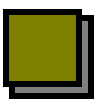
简单几何元件：



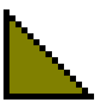
矩形元件是一个可以设定外框线和填充式样的矩形图形。填充式样可以设置为无填充，仅显示所绘的外框线；或无外框线，显示一个没有边框的矩形。



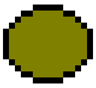
圆角矩形元件与矩形元件非常相似，只是有四个圆角。当元件被选择后，会出现一个额外的手柄，它是用来拖动改变圆角直径的.



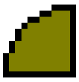
阴影元件与矩形元件非常相似，但它的右下方有一个阴影。这个元件通常设置为无填充，它可以在文本周围形成一个外框。



楔型元件是一个矩形框内的直角三角形。除了外框线和填充式样，它还有一个属性来指示它位于矩形框中的哪一个象限。



椭圆元件是一个可定义外框线和填充模式的椭圆形。填充模式可以设置为无填充，仅画一个椭圆线；又或者设置为无外框线，仅画一个椭圆形。



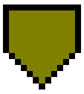
四分之一椭圆元件除了外框线和填充模式这两个属性外，还有一个属性就是图形的象限，它设定图形所占的位置。



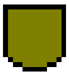
二分之一椭圆元件除了外框线和填充模式这两个属性外，还有一个属性就是图形的象限，它设定图形所占的位置，它也有四种位置。

这些元件的属性无需更多的解释，仅需要指出的是对于楔形元件，四分之一椭圆元件和二分之一椭圆元件中的象限属性可通过 Transform 菜单中的命令来更改。

油箱元件：



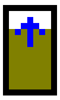
圆锥形油箱元件可定义外框线和填充模式。当选择了一个元件后，会出现两个手柄，通过拖动手柄可修改油箱的外形。



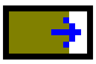
圆底油箱元件可定义外框线和填充模式。当选择了一个元件后，会出现两个手柄，通过拖动手柄可修改油箱的外形。

这些元件的属性无需更多解释。

基本液柱图形元件：

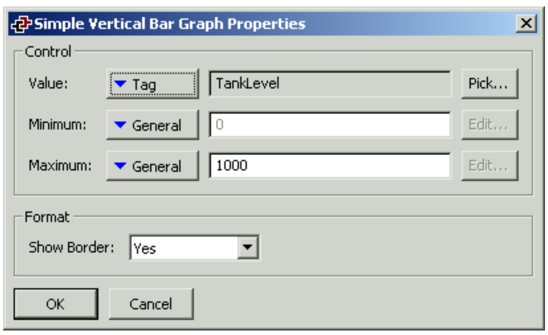


基本垂直液柱元件可以定义一个公式来表示液面的垂直高度，高度数值必须在最大值和最小值之间。另外还可以元件的外框线是否显示。



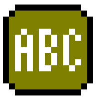
基本水平液柱元件可以定义一个公式来表示液面的水平长度位置，位置数值必须在最大值和最小值之间。另外还可以元件的外框线是否显示。

这些属性可以通过双击元件来访问：



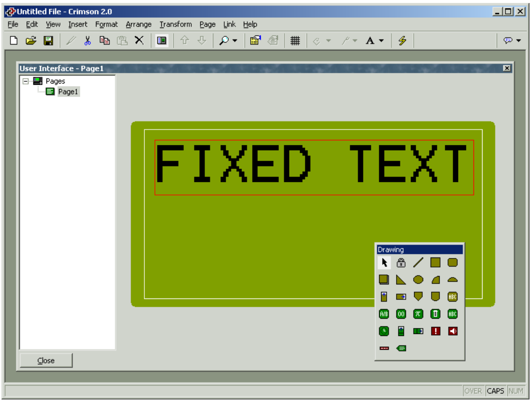
- Value 属性是用来设定显示值的，在上面的例子中元件被设置为显示油箱液位。
- Minimum 和 Maximum 属性是用来规定显示值的范围，在上面的例子中设定的范围是 0~1000。
- Show Border 属性是用来显示或隐藏元件的外框线。

固定文本元件：

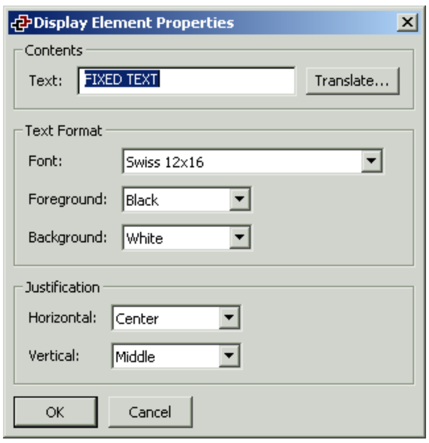


固定文本元件是用来添加不可变的文本。文本以设定的字体，颜色和对齐方式显示。文本可以被转译为其它语言。

当文本元件被创建后，会出现一个光标，指示输入文本：



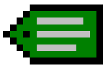
只有美国英语可以直接编辑，其它语言的文本必须通过属性对话框来编辑，属性对话框的访问可以首先选择元件，然后按 Alter+Enter 键，或从 Edit 菜单中选择 Properties 命令：



- TEXT 属性是用来输入显示文本的。就象上面提到的，美国英语文本可以直接在元件创建时编辑，或点击当前元件。

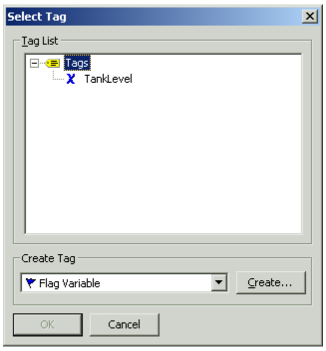
- Font 属性是用来设置字体的。这个属性可以使用工具栏中的字体键来更改，也可以用 Format 菜单中的命令来更改。
- Foreground 和 Background 属性是用来设置文本的颜色。显然，如果两个颜色设定为一样，文本将无法显示；如果 Background 背景颜色设置为无，将创建一个透明文本，文本元件下的元件都可以显示出来。
- Horizontal 和 Vertical Justification 属性是用来设置文本在方形元件框中的位置。这些属性都可以通过相应工具盒，或 Format 菜单来访问。

自动标签元件：



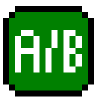
自动标签元件可用来选择一个数据标签，然后自动的在页面上放置相应标签的文本内容。例如：选择一个整数型标签，将会在页面上插入一个整数型文本元件。

这个图标是最常用的在页面上添加数据标签的工具，它首先显示一个如下对话框供你选择标签，然后会创建一个标签文本元件，标签文本元件共有五种，会在下节介绍。新的元件将使用数据标签的标注和格式属性，它是在创建标签时设置的。

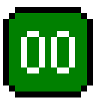


标签文本元件：

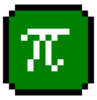
标签文本元件是用来以文本的形式显示和编辑数据标签的。首先它是用来显示标签的，显示的格式均按照数据标签窗口中格式表中的设置。如果你需要使用不同的显示格式，你可以直接在元件中更改格式。每一种类的数据标签都有一个标签文本元件与之对应：



标志型文本元件是用来显示真或伪状态的。



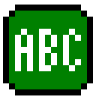
整数型文本元件是用来显示整数表达式的。



实数型文本元件是用来显示带浮点的实数表达式的。



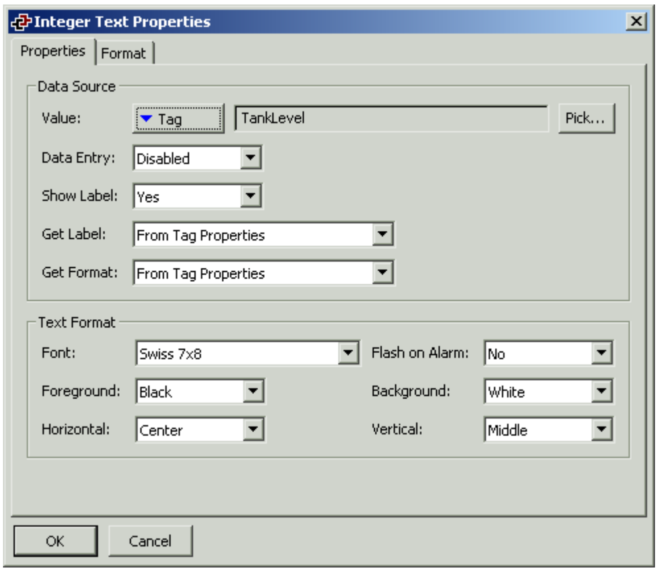
多重状态型文本元件是用来显示多重状态的。



字符串型文本元件是用来显示字符串表达式的。

标签文本元件的属性在两个表格中显示。

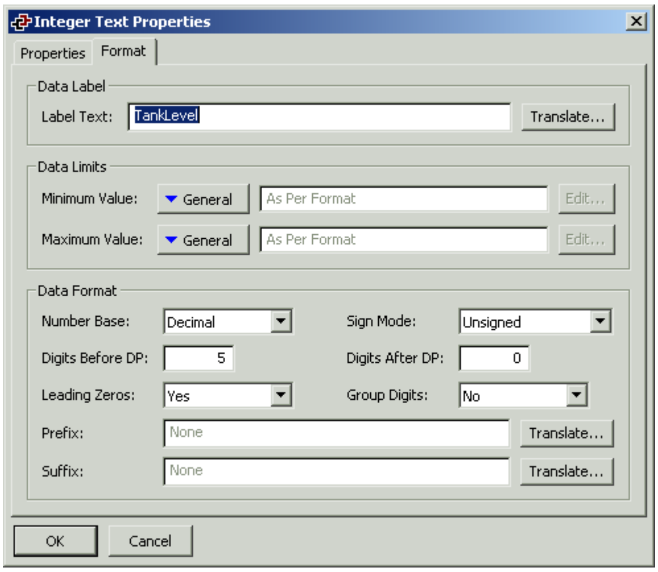
第一个表格的内容，五类元件基本相同：



- Value 属性用来设置该元件的数据来源。你可以选择一个数据标签，一个远端设备中寄存器，或是各种数据的综合表达式。数据的类型必须与元件类型相一致，如：一个整数型文本元件的数据值就不能设置为字符串型表达式。
- Data Entry 属性是用来设置该元件的数据值是否可以被操作员修改。如果数据输入需要设置为使能，首先数据值表达式必须是可更改的，例如：如果数据值是一个公式，那么数据输入将无法进行，被禁止。
- Show Label 属性可以设置是否需要将该元件的标注和数据一起显示。如果该属性设为是，在元件显示方框内，标注显示采用左对齐，而数据采用右对齐方式显示。如果该属性设为否，在元件显示方框内，数据显示采用水平对齐方式。注意：该属性也可以通过 Format 菜单中的 Field Label 命令来更改，如果没有选择元件这些命令将为新元件设置缺省值。

- Get Label 属性是用来设置从哪里获取标注文本。对于不同的数据值，该属性有不同的选项，如果数据值中是选择一个标签，你可以选择使用标签的标注作为缺省标注，或是在对话框的格式表中输入新标注；如果数据值中没有选择标签，你只能选择后者。
- Get Format 属性是用来设置从哪里获取格式信息。对于不同的数据值，该属性有不同的选项，如果数据值中是选择一个标签，你可以选择使用标签格式作为缺省格式，或是在对话框的格式表中设定新格式；如果数据值中没有选择标签，你只能选择第二个选项。
- Flash on Alarm 属性可用来设定，在当前显示的标签值处于报警状态时，该显示值是否闪烁。这项属性不适用于字符串文本元件，因为没有相应的报警值设定。
- 其余的属性是用来控制字体，颜色和对齐方式的，这些属性无须更多解释。

第二张表随元件的类型不同而不同，但都显示各类标签的格式信息。按照 Get Label 和 Get Format 属性的设置，格式表中不同的栏目将被使能。下面的例子是一个整数型文本元件的格式表：

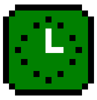


可以看到，所显示的属性与整数型标签的格式表完全相同。同样，其它类型元件的格式表也与相应类型标签的完全相同。详细信息可参考数据标签章节中解释。

编辑底层的标签：

如果你需要编辑标签文本元件的属性，可以左键双击该元件，或是右键点击该元件，再从弹出菜单中选取 Properties 命令。如果你需要直接编辑用于控制该元件的标签的属性，先右键点击该元件，再从弹出菜单中选取 Tag Details 命令。弹出的对话框将显示标签的数据表和格式表，你可以直接改变设置值。注意：通过这种方式更改的设置将影响所有与该标签有关联的元件。

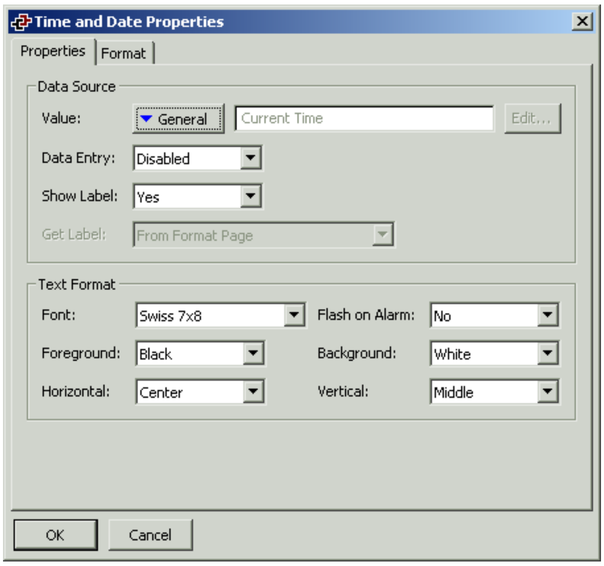
日期/时间元件：



日期/时间元件是用来显示日期和时间的，或显示包含时间/日期的表达式。它也可以用于修改表达式值或当前时间。

日期/时间元件的属性在两张表格中显示：

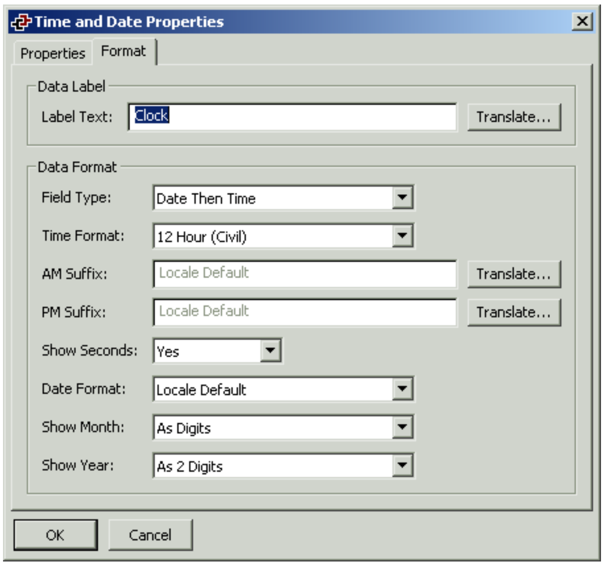
第一张表显示如下：



- Value 属性是用来设置需要显示的时间和日期。如果没有设定任何值，将会显示当前时间和日期。如果设定一个表达式，它的值将表示从 1997 年 1 月 1 日起，以秒为单位的时间。这些值通常是用 Time 和 Date 功能命令获得的，可参考功能命令章节。
- Data Entry 属性是用来设置是否可以让操作员在屏幕上修改 Value 属性中设置的时间值。如果 Value 属性中没有设置值，将是更改当前日期/时间。如果有设置值，表达式值必须是可更改的。例如：如果 Value 属性中是一个公式，数据将不能更改。

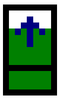
- 其它属性的功能与标签文本元件的相同。（似乎有些奇怪，此元件也具有 Get Label 和 Flash On Alarm 属性，因为 Value 属性也可以设置为一个标签，所以你可以决定是否需要标签的标注和报警状态。）

第二张表显示如下：

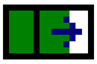


- Label Text 属性可用来设定元件的可选标注。
- Field Type 属性可设定时间/日期的显示模式，以及显示的顺序。
- Time Format 属性可设定以 12 小时或 24 小时形式显示时间，如设置该属性为 Local Default，Crimson 将按所选语言来显示相应时间格式。
- AM Suffix 和 PM Suffix 属性可设定在 12 小时时间模式下，上午和下午时间的显示后缀。如果你不设置该属性，Crimson 将使用缺省值。
- Show Second 属性可设定时间显示是否包含秒钟，或只显示小时和分钟。
- Date Format 属性可设定日期的显示顺序，如日，月，年。
- Show Month 可设置月份的显示模式，按 01 至 12 的数字显示，还是 Jan.至 Dec.的名字显示。
- Show Year 属性可设定日期显示是否包含年份，以及以多少位数字显示。

多功能液柱元件：

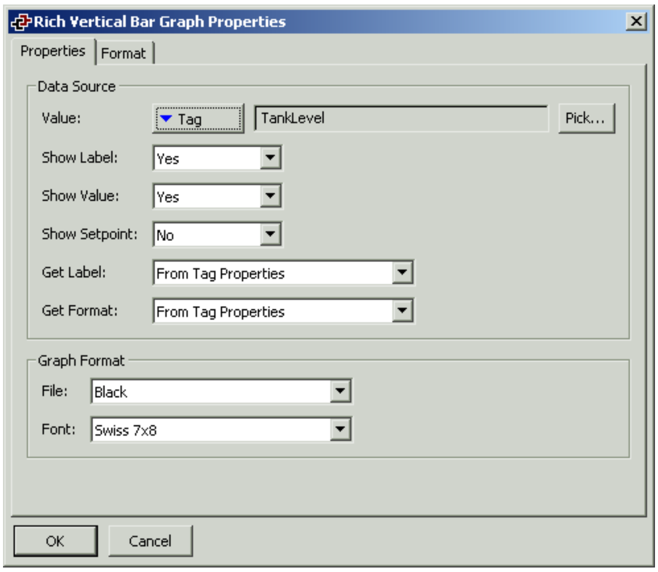


多功能垂直液柱元件可显示一个复杂的液柱图形，它可以包含标注，当前数据显示，以及设定点提示线。



多功能水平液柱元件可显示一个复杂的液柱图形，它可以包含标注，当前数据显示，以及设定点提示线。

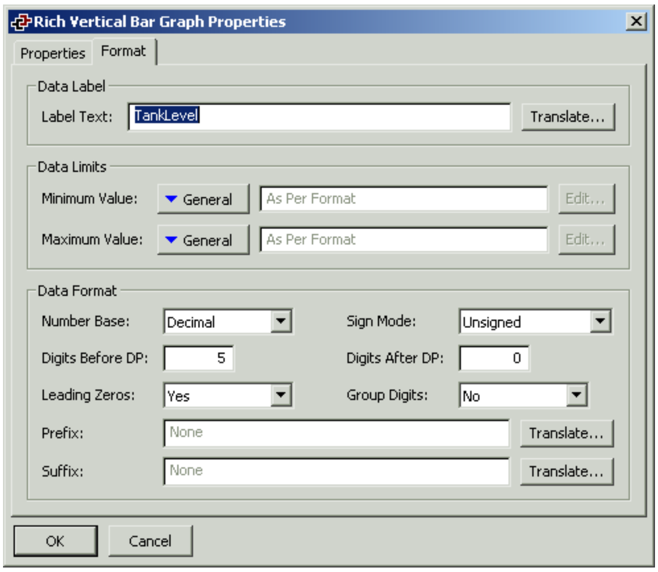
多功能元件的操作与标签文本元件的操作很类似，它可从数据标签中直接获取格式信息。与标签文本元件一样，两张表格包含元件的所有属性，第一张表格显示如下：



- Value 属性可用来定义需要显示的数据值。
- Show Label 属性可用来设定是否在液柱上显示文本标注。对于垂直掖柱，标注显示在图形底部；对于水平液柱，标注显示在左手侧；如果 Value 属性中是以标签赋值的，则标注可以从标签中获取，否则，必须直接在格式表中输入。
- Show Value 属性是用来设定是否在图形上显示当前数据值。如果 Value 属性中是以标签赋值的，则数据格式可以从标签中获取，否则，必须直接在格式表中输入。
- Show Setpoint 属性是用来设定是否在图形上显示设定点提示线。此选项只有在以标签方式赋值时才有效。
- Get Label 和 Get Format 属性与标签文本元件中的定义相同。如果 Show Value 属性设为否，则格式设置无效。

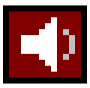
- Fill 属性是用来确定液柱图形的填充模式。如果你发现液注图形无法显示，首先检查此属性是否设置为无。
- Font 属性是用来设定图形中文本的字体，但首先如 Show Value 等属性必须设为使能。

第二张表格包含了标注和格式信息等属性：



上面显示的各种属性的功能与整数型标签的解释相同，可参考上面相应的章节。需要特别提出的是，为什么此元件必须输入最大值和最小值？即使数据是公式型的，无法在界面上输入，也必须输入最大值和最小值，这是因为如果没有定义数据范围，Crimson 将无法标定液柱的大小。

系统元件：



报警浏览器元件是为操作者提供的一个工具，通过它可以浏览和复位相应的报警信息。它通常与页面宽度一致，但会小于页面高度。



报警提示器元件以滚动显示的方式显示当前报警信息，它只占一行，通常与页面宽度一致，但操作者不能通过它对报警复位。

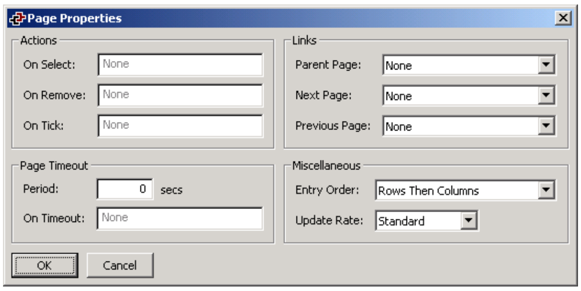


事件浏览器元件是为操作者提供的一个工具，通过它可以浏览和清除系统内部的事件记录信息。它通常与页面宽度一致，但会小于页面高度。

如果在系统的报警中你设置手动复位报警，你需要用一个页面来显示报警浏览器，以保证操作者可以对报警复位。你可以在其它页面上加入报警提示器，来提示操作者报警信息。同样，如果使用了事件记录，你也需要用一个页面来显示事件浏览器，以保证操作者可以浏览发生的事件。

定义页面属性:

通过 Page(页面)菜单可以看到，每一个页面都有一些属性：



- On Select 和 On Remove 属性是用来定义，当调入某一显示页面和退出某一显示页面时所执行的特定动作。可在编写动作和函数索引两个章节中找到所支持的动作。在本章的数据可用性一节中，可以了解到使用这些属性详细的时间定义。
- On Tick 属性是用来定义在当前页面显示时，每一秒钟都重复执行的设定动作。可在编写动作和函数索引两个章节中找到所支持的动作。但在这些属性定义的动作执行时，所需数据不可用，将跳过这些动作，直到下一秒钟再执行。
- Period 属性是一个以秒钟计的计时器，用来延时执行所设定的动作。
- On Timeout 属性用来设定计时器计时完成后，所执行的动作。
- Parent Page 是用来定义在当前页面按下 Exit 键时，哪一页将被显示。页面的选择也可以用下面的方法更改。
- Next Page 是用来定义在当前页面中，当光标位于最后一个输入项，按下 Next 键时，哪一页将被显示。页面的选择也可以用下面的方法更改。
- Previous Page 是用来定义在当前页面中，当光标位于第一个输入项，按下 Prev 键时，哪一页将被显示。页面的选择也可以用下面的方法更改。

如果你在一个页面中有许多个输入项，Next Page 和 Previous Page 可用来设定页面间的相互顺序。Crimson 将自动地定位光标位置，例如：当光标位于此页的第一个输入项时，按 Prev 键，所设定的前一页将显示，并且光标将位于此页的最后一个输入项。

- Entry Order 属性是用来设定光标在屏幕上输入项间的移动方向。也就是设定输入是按行顺序还是列顺序。

- Update Rate 属性用来设定页面中各项目的显示刷新频率。当增加刷新频率，会降低人机界面整体性能。建议最好使用缺省值。

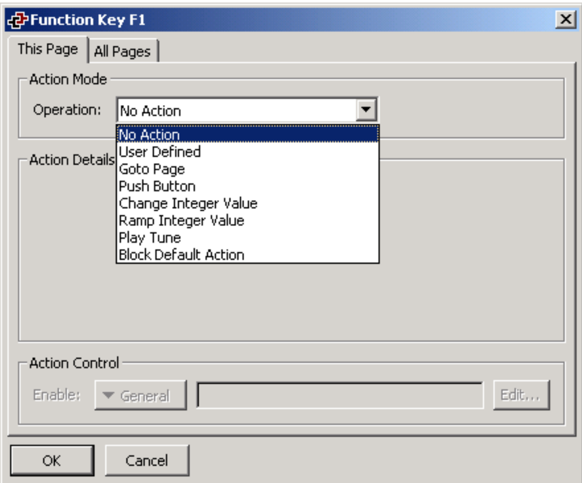
设定系统动作：

除了在页面属性中为每一页设定相应的功能，Crimson 提供一个方法，可在系统起动和每一秒钟都执行一些动作，而不论当前显示页为哪一页。这些动作可通过在用户界面窗口中左手窗中的 Pages 图标来访问。

设定按键功能：

前面的各章节中已经详细介绍了怎样通过 G3 把各种信息传达给操作者，而要完全完成用户界面的设置，剩下的就是怎样通过 G3 的按键与系统进行交互操作。

要设定一个按键的功能，首先选择视图缩放等级，使你可以看到要设置的按键。例如：你需要设置一个功能键，首先选择缩放等级 1 或 2，然后双击按键会显示如下：



你可以看到这个对话框有两个页表。第一页是用来设定在当前页中，按下按键将会执行什么动作。第二页是用来设定在所有页中，按下按键将会执行什么动作。第一页设定的动作称为本地动作，第二页设定的动作称为全局动作。按设定动作的不同按键的显示颜色也不同：



如果按键显示紫色，则设定为当页的本地动作。



如果按键显示绿色，则设定为全局动作。



如果按键显示蓝色，则同时设定为本地动作和全局动作。

当你已设定了一个按键，你可以用右键点击该按键，在弹出的菜单中选择 Make Global 或 Make Local 命令来更改动作类型，但当两种动作类型都已选时，此选项将不可用。

动作使能：

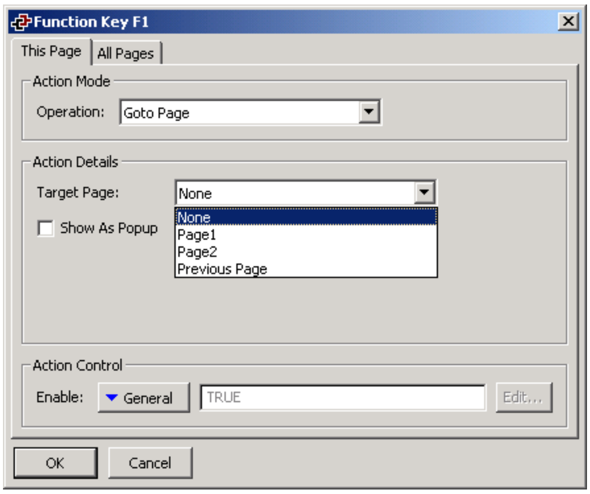
如果你需要为执行的动作设定一些条件，可在 Enable 栏中直接输入条件表达式。这个表达式可以是标志型标签的直接引用，也可以是在表达式书写章节中各种比较和逻辑运算的组合。如果你需要定义一个更复杂的逻辑组合，你可设定按键采用用户自定义模式，然后直接调用一个程序来完成所需逻辑。

动作描述：

下面的章节将详细描述各种类型的动作，当选定一种动作类型，动作对话框中的动作详情将显示不同的选项。

翻页动作：

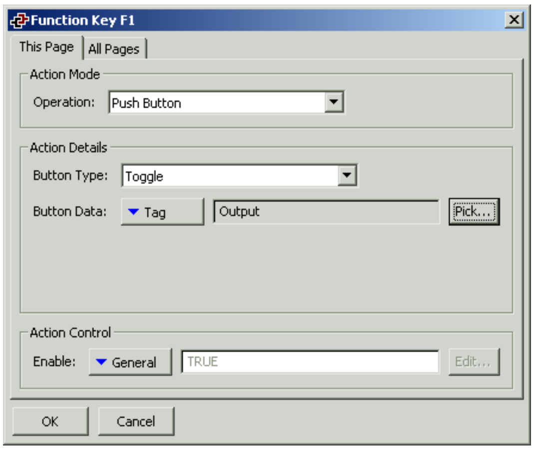
这个动作是用来指示 G3 显示一个新页面，选项如下显示：



- Target Page 属性是用来设定将要显示的目标页面。你可以指定需要显示的页面，或是选择前一页，来返回上次显示的页面。
- Show As Popup 选项将在当前页面的基础上，以弹出窗口的形式显示目标页面。目标页面弹出后，除了 Exit 键外，所有键均按弹出页面的设定功能执行，Exit 键是用来退出弹出页面的。

按钮动作：

该动作是用来模拟一个按钮的。其选项如下：



- 按钮类型属性是用来定义键的特性：

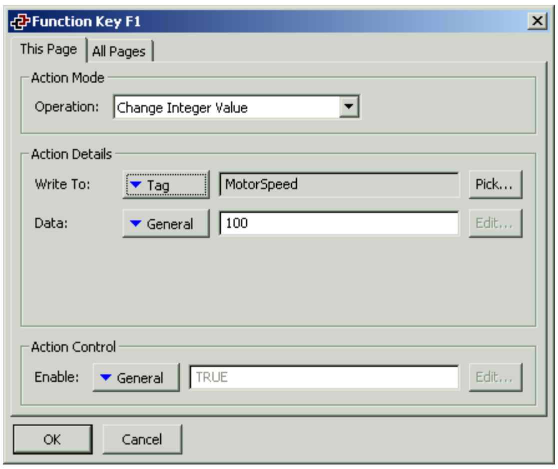
按钮类型	按钮动作
Toggle	每按一次，都会改变一次状态
Momentary	按下按键时置状态为 1； 释放按键时置状态为 0；
Turn On	按下按键时置状态为 1
Turn Off	按下按键时置状态为 0

- Button Data 属性是用来定义按键所改变的数据。

在上面的例子中，按键将转换 Output 标签值。

更改整数值动作：

该动作是用来向数据项写入整数值，其选项显示如下：

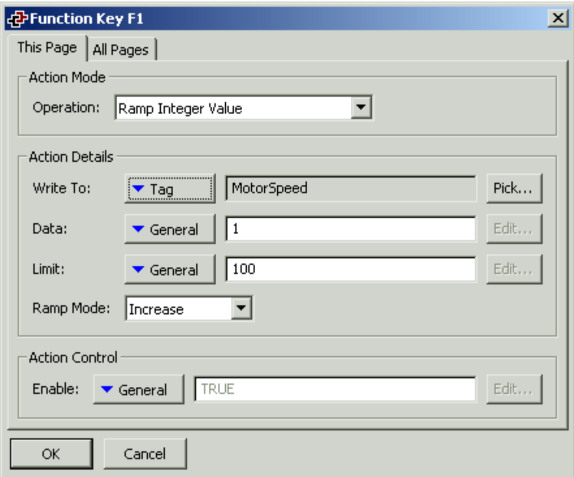


- Write To 属性是用来定义要更改的数据项。
- Data 属性是设定更改的值。

在上面点击例子中，该按键设定 MotorSpeed 标签为 100。

递增/减整数值动作：

该动作可增加或减少一个数据项值。其选项如下显示：

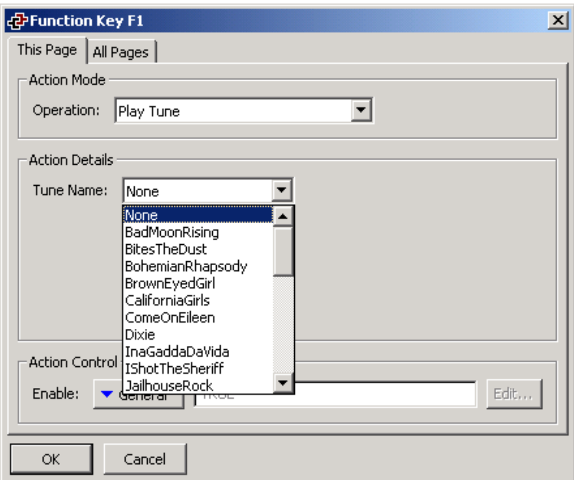


- Write To 属性定义要更改的数据项。
- Data 属性设定数值更改的幅度。
- Limit 属性定义数据的最大和最小值界限。
- Ramp Mode 属性设定是递增还是递减数值。

上面的例子中，保持按下按键，将递增 MotorSpeed 到 100，每次改变的幅度为 1。

播放乐曲动作：

该动作用 G3 内置的蜂鸣器播放乐曲：

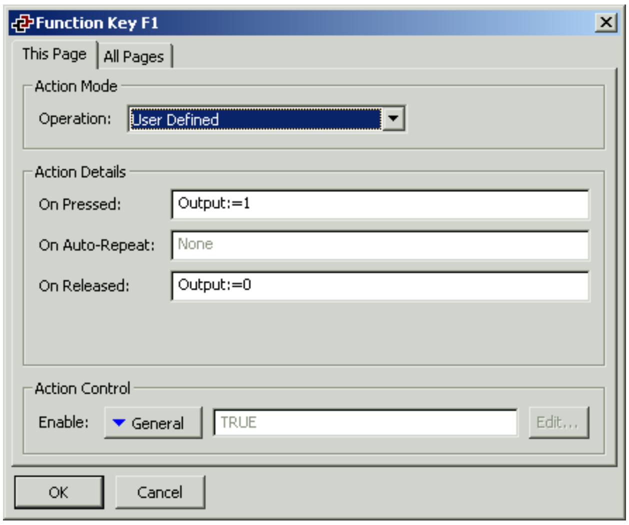


- Tune Name 可选择要播放的乐曲。

乐曲是使用电话音格式。可以用 PlayRTTTL()功能自定义乐曲。

用户自定义动作：

该动作可完成你需要完成的任何功能！选项如下：



- On Pressed 属性设定，当按键按下时，需要执行的动作。它可包括功能参考中所有的功能；或是动作书写中各种数据操作；或是运行一个程序。
- On Auto-Repeat 属性设定，当按键按下并保持时，需要执行的动作。该动作已包含起始动作和重复动作两部分，所以不必同时再设定 On Pressed 属性。它可包括功能参考中所有的功能；或是动作书写中各种数据操作；或是运行一个程序。
- On Released 属性设定，当按键释放时，需要执行的动作。它可包括功能参考中所有的功能；或是动作书写中各种数据操作；或是运行一个程序。

在上面的例子中，以用户自定义的方式完成了瞬动按钮的功能。

屏蔽缺省动作：

该动作不做任何实际动作，它的功能是一个阻断器，停止进一步的处理。例如：如果你设定 F1 键执行一个全局动作，但在某一页你不希望执行该动作，你可以在当页设定 F1 键为 Bolck Default Action，则在此页全局动作将不执行。

转换语言：

要设定一个键用来转换屏幕上的显示语言，首先设定按键为用户自定义模式，然后在 On Pressed 属性中输入 SetLanguage(n)，这里 n 为 1~8 之间的数值，代表显示的语言。显示的页面将按所选语言重新显示，包括各种文本：固定文本，标签标注，标签格式信息等。其它的页面也按同样的语言显示。

高级设置：

下面的章节将描述与键盘动作有关的高级设置。

动作处理：

当一个按键被按下或释放，Crimson 在决定处理这个事件之前，首先会检查一个规定的过程，如果此时有一些动作所产生的状态正在被执行，那么检查过程将会被停止，那么这个按键的动作就无法执行了。

相关的动作有：

1. 如果一个显示元件正处于与用户交互，它将调用键的功能。一个激活的数据输入元件将占用 Raise，Lower，Exit，和 Enter 键，以及其它与操作有关的键。例如：整数输入将占用数字键。
2. 如果一个显示元件正处于与用户交互，Next 或 Prev 键被按下，Crimson 将寻找下一个或前一个与用户交互的元件，如果元件存在，则会被激活，按键也会被占用。
3. 如果设定了一个本地动作，并且动作被执行，该键被占用。
4. 如果设定了一个全局动作，并且动作被执行，该键被占用。
5. 如果按键一直处于未被占用，缺省动作可被执行：

事件	**动作**
按比例 Next 键	显示下一页，如果已定义。
按比例 Prev 键	显示前一页，如果已定义。
按比例 Exit 键	显示上一页，如果已定义。
按比例 Menu 键	显示第一页。
按比例 Mute 键	报警蜂鸣器静音。

如上面所提到的，当设定一个键用于全局或本地动作，恰好某一页中不需要它执行，此时如屏蔽缺省动作，就非常的有用，可有效的停止此动作。

数据可用性：

Crimson 的通讯基础结构规定，只读取当前页面所需的数据。这也就是说，当一个页面第一次被选中时，一些数据可能是不可用。对于一个显示元件，这没有太大影响，元件可显示一个不确定状态（通常显示一条虚线），直到数据可用。对于动作，过程就比较复杂。

例如：假设我们设定一个本地动作为增加电机转速至 50rpm。如果电机转速在前一页没有被使用到，当此页第一次显示时，Crimson 并不知道当前电机转速，也就无法写入新值，当操作者在数据不可用的状态下执行该动作时，G3 会显示“NOT READY”信息，直到操作者释放按键。操作者可以稍等片刻再进行操作。实际上通讯的刷新是非常快的，即使你的手非常快，也很难出现上述情况。但必须对这种情况解释清楚。

但如果动作是用页面的 On Select 属性设定的，情况又稍稍复杂一些，这时 Crimson 会等待 30 秒钟，如果数据还是不能到达，动作将不执行，并显示”TIMEOUT”信息。计时超出的方法可以避免因通讯中断而造成的问题。

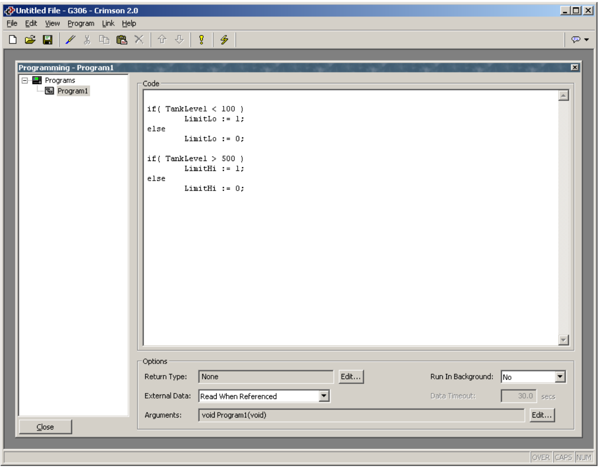
Edict 使用者须知

Edict97 的使用者必须注意：

- 页面不再有文本层和图形层之分，所有元件都是图形化的。也就是说，页面格式的概念已没有了。
- 页面种类已由系统元件代替。Edict 会使用整个页面显示报警浏览器，而系统元件可以按需要使用页面。
- 动作通过双击按键来编辑，代替了全局和本地事件图。如果应用中需要多行的功能或命令，可用程序来完成所需逻辑。
- 一些事件如通讯更新完成和每秒钟计时已经废除了。需要使用此类事件的动作，可用其它方法实现。例如：通讯更新完成事件通常用于在两个设备间传送数据，现在可用通讯窗口中的协议转换功能来实现。这些时间经常被滥用，以至于数据库文件过于复杂。
- Edict 通常每秒钟刷新页面 2~5 次，而 Crimson 一般每 100mS 刷新一次页面，操作者在数据输入时会更流畅。

编写程序

在本手册的前面章节中，我们了解了怎样运用按键和更改数据标签来执行动作，以完成各种操作。但如果要完成一个非常复杂的动作，在一行表达式中无法完成；或需要一个复杂的逻辑运算，你就需要使用在主页面中的编程图标，来创建一个程序。在大多数的应用中是不需要程序的，你可以选择跳过这一章节。



使用程序列表：

要创建、重命名和删除程序，可以先点击编程窗口的左手栏，再使用 Program 菜单中的各种命令来完成。同样也可以右键点击相应程序，从弹出菜单中选取命令。

要选择一个程序，可以点击列表中的程序名，也可以使用工具栏中上/下箭头；再或者使用 Alt+Left 和 Alt+Right 组合键在列表中移动。这些键可在任何窗口中使用。

编辑程序：

要编辑程序，可直接在编程窗口的右手栏中编辑操作。当完成编辑后，按 Ctrl+T 或在 Program 菜单中选择 Translate 命令即可，它将读去程序并检查错误。发现错误后，会显示一个对话框，同时光标会移到错误位置。如果没有发现错误，会显示一个对话框提示没有错误。程序会被编译成为 Crimson 内部代码供人机界面执行。

程序属性：

在右手框的底部有一些栏目是用来编辑程序属性：

- Return Type 属性可设置该程序只是完成一系列动作或者是进行计算后返回计算值。返回计算值的程序将在下面详细介绍。

- Run In Background 属性是设置 Crimson 是否需要等待程序执行完成然后再继续执行任务。例如：假设该属性设置为 No，当按下按键去运行一个程序，此时会中断显示刷新直到程序运行完成（由于大多数程序执行的时间非常短，一般它是注意不到。）。如果该属性设置为 Yes，显示刷新会继续，程序以一个较低优先级在后台运行。在后台每次只能运行一个程序，多个程序会按队列顺序进行。注意：有返回值的程序不能在后台运行，这样引用程序的动作将无法得到可用数据。
- External Data 和 Timeout 属性用来控制程序怎样就外部数据的引用与 Crimson 的基础通讯机制进行交互。可以回想到 Crimson 只读取正在使用的数据，这个属性就是以这条原则为基础来控制相应程序的数据交互。

模式	特性
Read When Referenced	无论何时程序被调用，程序所包含的外部数据将被列入通讯扫描中。如果程序在一个显示页面中被调用，在页面显示时，数据将被读取；如程序在一个全局动作或触发器中被调用，数据将始终被读取。这是缺省模式，可适用于所有程序，除非外部数据的数量非常大。
Read Always	程序所引用的外部数据将始终被读，无论程序是否被引用。这样程序随时可以运行，不会出现“NOT READY”信息，即使程序第一次被引用。此模式的缺点是，如程序中引用大量数据，通讯性能会降低很多。
Read When Executed	只有在程序运行时，外部数据才会被读取，程序会等待一段由 Timeout 属性定义的时间段，来等待数据可用。如果数据不能读到，可能是由于设备离线，程序将不会执行。此模式通常用于一些包含大量数据的全局引用程序，它会明显减慢通讯刷新。
Read but Run Anyway	外部数据的处理与 Read Always 模式相同，但无论数据是否成功读取，程序都会运行，不会出现 NOT READY”信息，但如果设备离线将无法保证程序的数据是有效的。

加入注释：

有两种方法可以在程序中加入注释，首先可以使用“//”后接单行注释语句。其次可以使用/*后接单行或多行的注释，然后以*/结尾。下面是两种注释的例子：

```
// This is a single-line comment
/* This is line 1 of comment
   This is line 2 of comment
   This is line 3 of comment  */
```

单行注释也可以放置在程序代码后面。

返回数据值：

如上面所提到的，程序可以返回数据值。这种程序可以被其它程序或表达式调用。例如：如你希望按一个电机的状态产生一个复杂的编码并用返回值来指示当前状态，你可以创建一个程序并返回整数值：

```
if ( MotorRunning )
        return 1;
else {
        if ( MotorTooHot )
                return 2;
        if ( MotorTooCold )
                return 3;
        return 0;
        }
```

你可以设置一个多重状态公式型变量来调用此程序，并用标签的格式表来定义不同的状态名。调用是通过设定标签的 Value 属性为 Name()完成的，Name 是指程序名。圆括号用来指示函数调用，不能省略。

注意陷阱：

对于用程序返回数据值，必须多加练习。在实际中，程序应避免使用过长的循环语句，以及执行与调用没有上下关联的动作。例如：上面的程序是用来完成自动翻页功能，但如果你在数据记录功能中调用该程序，你能想象结果会怎样！所以，程序返回值一定要简单，并且要符合上下动作关系。否则，就使用基本的数学公式和 If 语句；来完成。

编程技巧：

下面的章节中将对 Crimson 所支持的程序结构作一个概述，基本的使用句法与 C 语言的相同。注意，这里并不是教大家怎样成为程序员或者是掌握 C 语言的细节。这些已超过手册的范围。我们的目的是对可用的函数有一个概述，感兴趣的用户可进一步研究。

多个动作：

最简单的程序是有一些动作组成，每一个动作占据单独一行，以分号结束。在编写动作章节中的所有动作都可以使用。象下面这个简单程序可以把多个动作集合起来，用一个动作来表达，增强了可读性。

下面的例子设置一些变量，并更改显示页面：

```
Motor1 := 0;
Motor2 := 1;
Motor3 := 0;

GotoPage(Page1);
```

动作会依此执行，然后程序返回调用处。

IF 语句：

IF 语句在程序中是用来作判断的。它包括圆括号中的条件和随后的一个或多个动作。当条件满足时执行动作。如果定义了多个动作，必须每个动作占据一行，同时用大括号括起。一个可选的 else 语句可用来定义当条件不满足时所执行的动作。

下面的例子是一个单独动作的 IF 语句：

```
if ( TankFull )
        StartPump := 1 ;
```

下面的例子是一个两个动作的 IF 语句：

```
if ( TankEmpty )  {
        StartPump := 0 ;
        OpenValve := 1 ;
        }
```

下面的例子是一个具有 else 从句的 IF 语句：

```
if ( MotorHot )
        StartFan := 1 ;
else
        StartFan := 0 ;
```

注意：在 if 和 else 语句后的一组动作，必须用大括号括起，如果省略括号，Crimson 将会误解基于条件的动作，虽然动作间是有分搁符，但还是无法区分哪些动作在条件范围内。

Switch 语句：

Switch 语句是将一个整数变量与一些可能的常数相比较,当数值相等时执行相应的动作。正确的句法支持多个选项，在大多数应用中，下面的例子是比较典型的。

下面的例子是按 MotorIndex 标签的值来启动相应的电机：

```
switch ( MotorIndex ) {

        case 1 :
                MotorA :=1;
                break;
        case 2 :
        case 3 :
                MotorB :=1;
                break;
        case 4 :
                MotorC :=1;
                break;
        default :
                MotorD :=1;
                break;
        }
```

MotorIndex 等于 1 时启动 A 电机；等于 2 或 3 时启动 B 电机；等于 4 时启动 C 电机；等于其它值时启动 D 电机。使用的句法是，Case 语句要用大括号括；每一个条件块以 Break 结束；连续的两个 case 语句表示同一个条件块；可选的 default 语起句是在数据都不匹配时，指定一个缺省动作。（如果这个语句看起来有些复杂，可以用多个 if 语句代替它，但有时性能有所降低，可读性较差。）

局部变量:

程序经常使用变量来存储过程值，或控制循环结构。你不需要定义标签来存储这些值，你可以使用下面的句法来定义局部变量来处理：

```
int        a;     // Declare local integer   'a'
float      b;     // Declare local real      'b'
cstring    c;     // Declare local string    'c'
```

局部变量可以在定义时进行初始化赋值，可在变量名:= 数值的格式，没有初始化赋值的变量被置为 0 或空字符串。

注意：局部变量只是在局部范围和局部时间有效，它不能被外部引用。它在多次调用之间不能保持数值。如果程序按递归的方式调用，每次调用都有自己的变量。

循环结构：

有三种不同的循环结构在条件满足时执行给定的动作。While 循环在动作执行前检查条件；Do 循环在动作执行后检查条件；For 循环是一种快速定义 while 循环的方法，可将三个基本要素在一个语句中定义。

由于编程的错误会造成循环无法终结，我们必须特别注意循环的使用。基于不同的程序调用，它会引起用户功能和通讯的崩溃。太多的循环次数也会引起下层系统的性能降低。

While 循环：

这类的循环在 while 语句中的条件成立时，重复执行后面的动作。如果条件始终不满足，动作始终不执行。在检查循环条件之前，循环动作不执行。如果你需要执行多个动作，必须以大括号括起，这与 if 语句相同。下面的例子中，首先初始化两个局部变量，用第一个变量构成循环，读取数组中的数据，将前十个数据累加，最后向调用者返回累加值。

```
int     i := 0, t := 0;

while ( i < 10 )  {
        t := t + Data [i] ;
        i := i + 1 ;
        }
return t ;
```

下面的例子与上面的程序相同，但是以简化的格式。由于仅有一个动作，大括号可以省略了：

```
Int      i := 0, t := 0;
while ( I < 10 )
         t +=  Data [i++] ;
return          t ;
```

For 循环：

可以注意到上面的 while 循环有四个要素：

1. 循环变量的初始化。
2. 检查循环条件，确定循环是否进行。
3. 执行循环动作。
4. 更改循环变量。

For 循环将 1，2，4 要素合并为一个语句，要素 3 紧跟其后。这个句法与 Basic 语言中的 For-Next 循环很相似。使用这种语句，上面的例子可以写成：

```
int      i , t ;
for ( i:=t:=0; i<10; i++)
         t +=  Data[i] ;
return t ;
```

可以注意到，for 语句有三个独立要素，以分号分隔。第一个是起始，它在第一次执行循环是执行；第二个是条件，它确定是否执行循环动作；第三个是归纳，它更改循环变量，进入下一次循环。当有多个循环动作时，一定使用大括号！

Do 循环：

这种类型的循环与 while 循环很类似，只是循环条件检查放在循环尾部。也就是说循环至少执行一次。用 do 循环可改写上面的例子：

```
int  i:=0, t:=0;
do {
        t += Data [i]
        } while ( ++i < 10 );
return t ;
```

循环控制：

有两个语句可以用于循环中，一个是 break 语句，可用来提前中断循环块。另外一个是 continue 语句，它可用来跳过剩余的循环块内的动作，而进入下一次循环。这两个语句必须和 if 条件语句合用才有意义。下面的例子中，当另一个程序的返回值为真时，提前中断循环：

```
for  ( i:=0; i<10; i++ ) {
        if ( LoopAbort () )
                break;
        LoopBody ();
        }
```

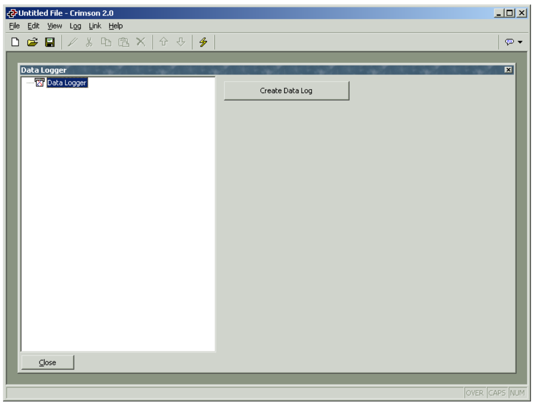
Edict 用户须知

Edict97 的用户必须注意：

- Crimson 支持的局部变量，是通过程序内部的 C 语言方式声明来完成的，而不是 Edict 中的局部变量表。Crimson 在堆栈中保持局部变量，所以可以以递归的方式引用。但不能在两个程序间引用。
- Crimson 不支持 Dispatch 功能，程序是在前台或后台运行，是有属性决定，而不是调用的方式。
- Crimson 按 C 语言句法调用程序，老的句法也支持，但不必用 Run 功能。调用程序返回值需用新的句法，Edict 中的 RunInteger 等此类功能已不使用了。
- 程序在 Crimson 的运行速度比 Edict 中要快。

设置数据记录

现在你已经设置了一些应用中的核心功能，接下来要了解怎样利用 Crimson 的数据记录器将一些标签数据记录到 CF 卡上。用这种方法记录的数据是工业标准的逗号分隔变量文件（CSV），可以简便输入到 Excel 等应用软件中。要设置数据记录，在主界面中选择数据记录器图标：



创建数据记录:

你可以使用 Create Data Log 按键来创建多个数据记录，由于每个数据记录没有限制数据标签的数量，多数应用中只有一个数据记录。然而，一个数据记录都按同一种属性处理，如同样采样速度。当你需要用不同的采样速度时，可用不同的数据记录来处理。

使用记录列表：

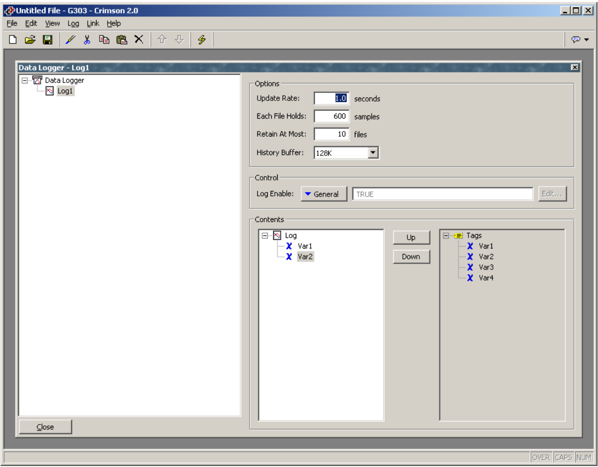
要重命名和删除数据记录，可以先点击数据记录器窗口的左手栏，再使用 Log 菜单中的各种命令来完成。同样也可以右键点击相应数据记录，从弹出菜单中选取命令。

（注意：数据记录的文件名必须小于或等于 8 个字符。因为这个文件名同样用来设定记录文件目录的名称。Crimson 不能操作不符合 FAT-style 8.3 的命名。）

要选择一个记录，可以点击列表中的记录名，也可以使用工具栏中上/下箭头；再或者使用 Alt+Left 和 Alt+Right 组合键在列表中移动。这些键可在任何窗口中使用。

数据记录属性：

每个数据记录都有如下属性：



- Update Rate 属性用来设定数据采样的速度，最快的速度是每秒一次。但注意这样的速度会产生大量的数据！记录中所有数据标签都以同样的速采集。
- Each File Hold 属性设定每一个记录文件包含多少个采样。当采样数量达到后，会以不同的文件名建立新的记录文件。通常这个数量设定每个记录文件的总记录时间。例如，上面的设定是一个文件记录一天。
- Retain At Most 属性设定在 CF 卡总存储多少个文件。这个数量的设置应保证在文件删除前，所有数据已经提取了。上面的例子是保持一周的数据记录。
- Log Enable 属性用来控制记录的起动和停止，记录的缺省设置是起动。
- Contents 属性设置要记录哪些标签。第一个列表显示已选择的标签，第二个列表显示数据库文件中可用的标签。双击右手列表中的标签，可将它加入记录中；在左手列表中选择标签后，用 Del 键可删除已选标签。UP 和 Down 键可在列表中移动光标。

记录文件存储：

上面提到，数据记录的文件是存储在人机界面中的 CF 卡上。文件是存放在以 Data log 名字命名的子目录中，整个目录存放在 LOGS 根目录中。目录中的文件是以记录开始时的时间和日期命名的。如果每个文件包含一个小时以上的信息，文件将命名为 YYMMDDhh.CSV，这里 YY 代表年，MM 代表月份，DD 代表日期，HH 代表小时。如果文件包含小于一个小时的数据，文件名将是 MMDDhhmm.CSV，前六个字符的含义与上面相同，mm 代表起始的分钟。这个规则可保证每一个文件名是唯一的。

数据记录过程：

Crimson 的数据记录分两个单独的过程。第一是按采样频率采集数据，将采样的数据存储到 G3 人机界面中的 RAM 缓冲区中；第二步是每两分钟将缓冲区中的数据写入 CF 卡中。这种方式有一些优点：

- 向 CF 卡写入数据是以两分钟为界，2，4，6 分钟，至 1 小时等。这就是说如果你的 CF 卡支持热拔插功能，你可以等待下一次写操作，当写操作完成后，CF 卡写入的 LED 指示灯停止闪烁，你就有两分钟时间拔出 CF 卡并无需担心数据中断。四分钟之内插回一个新 CF 卡，数据将继续记录，并保持完整。
- 这种向 CF 卡中写入数据的方式具有很大的优点，可以避免在每一个采样点都持续向卡中写数据。对于采样速度非常高的记录，CF 卡的带宽也不支持不经缓冲处理直接写入。

注意：数据以每两分钟为单位写入 CF 卡，如果系统掉电，最多会丢失两分钟的数据。另外，在写入过程中系统掉电，CF 卡会损坏，但不是永久性的。G3 内部使用一个日志记录，将所写数据记录在可保持内存中。当发现因掉电而引起写操作中断，再上电时，会重复写操作，并修复 CF 卡。

当你需要拔出带有数据记录的 CF 卡时，必须首先观察写操作的 LED 指示灯，当写操作停止后，才可以断电。如果你不能确定操作是否正确，可重新上电，再进行一个写操作后，按正确程序断电，卡就可以安全，完整的取出了。

由于拔出 CF 卡的过程比较复杂，Crimson 提供了其它两种方法访问记录文件，这样就不必 CF 卡拔出了。具体方法如下：

访问数据记录文件：

有两种访问数据记录文件的方法：

- 正如在本手册开始时介绍的，可以将 CF 卡安装成为计算机的一个驱动器。这样记录文件可以直接用 Windows Explore 复制访问。注意：使用这种方法时，推荐使用 Windows 2000 或以上版本，较早的 Windows 版本会错误的锁定 CF 卡，使数据记录无法进行。
- 另一种较好的方法是使用下一章介绍的 Web Server 功能。当使能 Web Server 功能，记录文件可以通过人机界面的以太网口来访问，可使用网络浏览器，如 Microsoft Internet Explorer，或使用 Crimson 所提供的 WebSync 应用程序，完成自动的数据交换。

使用 **WebSync**：

WebSync 应用程序，是在 Crimson 软件安装时存储在指定目录中，它可将 PC 上的一个目录与人机界面上的数据记录文件同步联接，你可以在 Windows 的计划表中创建一个项目，定期的运行这个应用程序；或者也可以在命令提示符下，执行 WebSync 程序，以完成自动数据轮询。你还可以将 WebSync 在服务器上运行，这样局域网的指定用户就可以自动获取记录文件。

WebSync 句法：

WebSync 的调用使用以下句法：

Websync {switch} <hostname>

这里<hostname>代表人机界面的 IP 地址。

Switch 选项：

- -terse 用来抑制过程信息。
- -poll <n>设定每 n 分钟轮询一次。
- -path<dir> 设定保存记录文件的目录。

用法示例：

websync -poll 10 –path c:\logs 192.9.200.52

该指令将从 IP 地址为 192.9.200.52 的屏中读取数据记录，并存储在 c:\logs 目录中的子目录中。Websync 会每 10 分钟轮询一次。轮询间隔时间必须被设置远小于采样速率乘以采样点数乘以文件数。如果这个条件满足，PC 目录中会保留所有的记录文件。

Edict 用户须知

Edict97 用户必须注意：

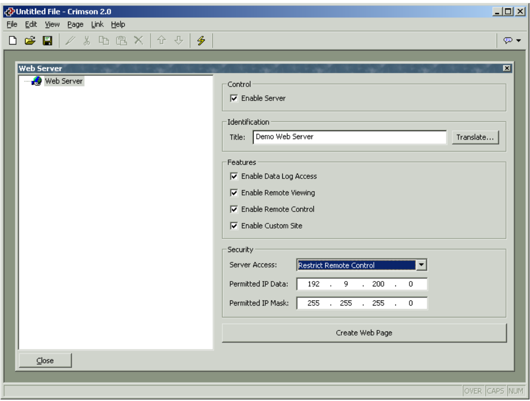
- Crimson 的数据记录文件记录了每一个采样点的时间和日期，不必象 Edict 中在掉电期间存储“空”值。在 G3 掉电时，数据记录文件只是简单的有一段空白。

设置 Web Server

Crimson 的 Web Server 功能可以通过 G3 的以太网口，收发各种数据，可以远程访问诊断信息，以及数据记录器中的记录。设定 Web Server 首先在主页面中选择 Web Server 图标。

Web Server 属性：

Web Server 的属性有：

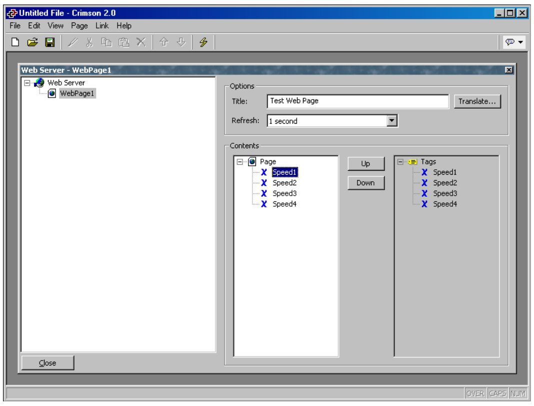


- Enable Server 属性用来使能或禁用 Web Server 功能。当 Web Server 使能，人机屏会监控 80 通讯口的访问请求，并按请求应答。如果 Web Server 禁用，与 通讯口的连接会被拒绝。注意：要使 Web Server 正常工作，首先应在通讯窗口中对以太网口使能。
- Title 属性是设定显示在 Web Server 菜单上的名称。在一个网络中不同的终端设备应有不同的名称，这样才能够保证正确的访问。
- Enable Data Log Access 属性用来使能或禁用通过 Web Server 访问数据记录器所记录的文件。显然，如果使用 WebSync 应用程序将记录文件复制到 PC 上，该属性必须使能。
- Enable Remote View 属性用来使能或禁用通过 Web Server 浏览当前 G3 界面的显示内容。这个功能在远程诊断时非常有用，操作者就象站在操作屏前一样操作和控制。
- Enable Remote Control 属性是在使用远程浏览功能时，使能或禁用它的选项功能，就是通过网页浏览器模拟按键功能，完成远程操作和控制。虽然这这个功能非常有用，但必须使用相应的安全设置参数，以防止非法的操作设备。如果设备接入 Internet，必须使用外部防火墙以确保安全。

- Enable Custom Site 属性用来使能或禁用通过 Web Server 访问存储在 CF 卡中 WEB 目录下的文件。此功能将在下面详细讨论。
- Security 属性用来限制通过 Web Server 方式访问的远端设备，只有当远端设备的 IP 地址和掩码符合所设置的数值时，才允许访问。可以选择限制所有访问，或限制远程控制。使用外部防火墙以防止非法的访问是用户的责任，因为通过 IP 地址筛选的方法是可以被高级黑客技巧突破，红狮公司不确保其安全性。

添加网页

除上面的功能以外，Web Server 还支持显示普通网页，每个网页都可以包含预设的数据标签。这些网页可以用 Web Server 下面的 Create Web Page 按键来创建，并存储在与显示页面和数据记录一样的列表中。



每个页面都有如下属性：

- Title 属性是设定显示在 Web 浏览器菜单上的名称。此名称是可翻译的，当前设定是美国英语文本。
- Refresh 属性是设置网页浏览器是否自动刷新页面内容。更新频率从 1 秒至 8 秒可选。注意：页面的闪烁次数是由页面数据包和机器性能决定的，更新频率无法消除闪烁。
- Content 属性用来选择哪些数据标签显示在页面上。第一列列表是已经选择的标签；第二列列表是可选的标签。双击右手列表中的标签，可将它加入记录中；在左手列表中选择标签后，用 Del 键可删除已选标签。UP 和 Down 键可在列表中移动光标。

使用自定义网页：

虽然使用标准的网页可快速方便的访问屏中数据，但你可能发现你不能更改它的格式。如果要构建一个贴合应用的网页，你可以使用人机界面中的自定义网页功能，用第三方软件，如 HTML 编辑器，来创建全新的网页，并插入特殊的序列项，最后将完成的网页存储到 CF 卡上，你就可以通过 Web Server 功能访问了。

创建网址：

网址必须使用你的浏览器所支持的 HTML 格式，不能使用 ASP，CGI 或其它格式。网页的文件名和其它图形的命名必须按照老的 8.3 命名惯例。这也就是说，文件的扩展名必须是 HTM，而不是 HTML；是 JPG，而不是 JPEG 等。同样，文件名要少于或等于 8 个字符，名字不分大小写。你可以使用任何目录结构，只要目录名称按照 8.3 命名惯例，并且不分大小写。

嵌入数据：

要在网页中插入标签数据，只需要插入序列项 [[N]]，N 为标签的索引数字。在数据标签窗口选择了标签后，索引数字会显示在下方的状态栏中。其大小与创建的顺序有关。当包含序列项的网页被调用，标签的当前值将显示在序列项位置，格式按照标签属性设定。

展开网址：

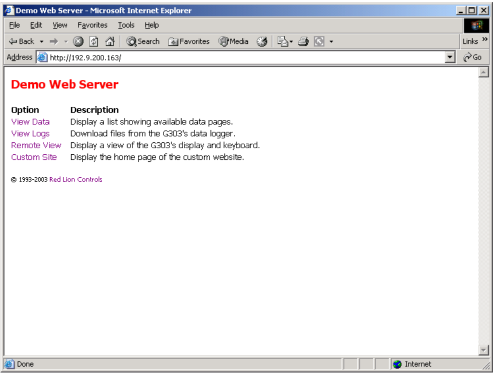
要展开自定义的网址，只需要将它复制到 CF 卡的 \WEB 目录中即可。你可以将 CF 卡作为 PC 的一个驱动器，来复制文件，就像手册前面所介绍的；也可以直接将 CF 卡连接到 PC 机的 CF 卡读写器。Enable Custom Site 属性必须为使能，自定义网址就会出现在 Web Server 菜单中，当选中网址后，\WEB 目录下的 DEFAULT.HTM 文件就显示出来了，从这一页起，按设定的链接显示各页。

CF 卡的访问：

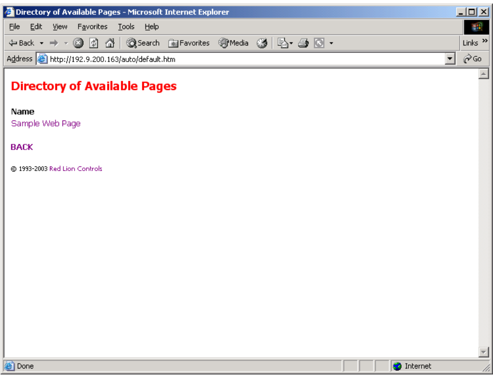
注意：要使用自定义页面或访问人机界面的数据记录，Web Server 必须访问 CF 卡。如果你将 CF 卡安装成为 PC 机的一个驱动器，并且执行写操作，你只有等一分钟后或 PC 机对 CF 卡解锁后才能通过人机界面访问。如果你使用 Windows 2000 以前的版本来执行这个操作，你会发现无论你是否执行读写操作，PC 机在安装驱动器时会锁定 CF 卡，只有等一分钟后，才能解锁。

Web Server 样板：

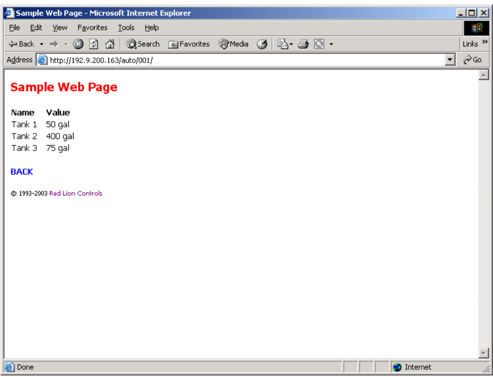
下面的图片显示的是 Web Server 的主菜单：



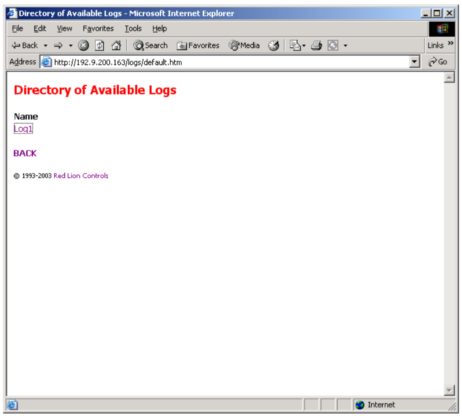
下面的图片显示的是标准网页列表：



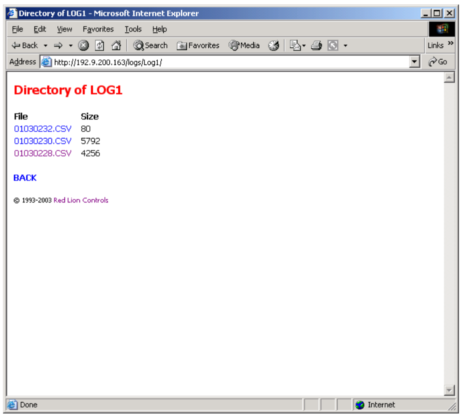
下面的图片显示的是标准网页包含的三个标签：



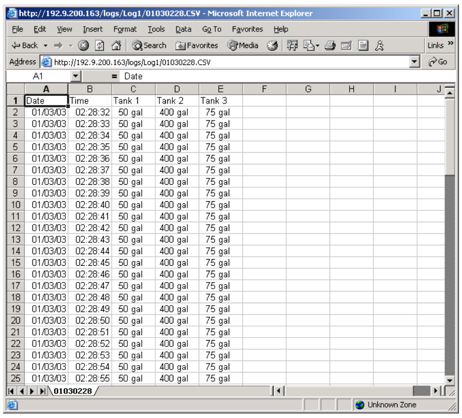
下面的图片显示的是数据记录菜单：



下面的图片显示的是选定的数据记录的内容：

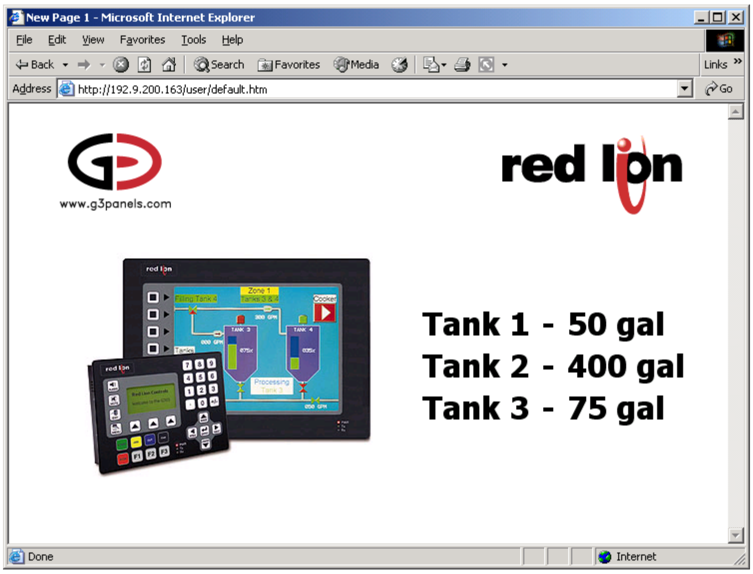


下面的图片显示的是选定的数据记录文件的内容：

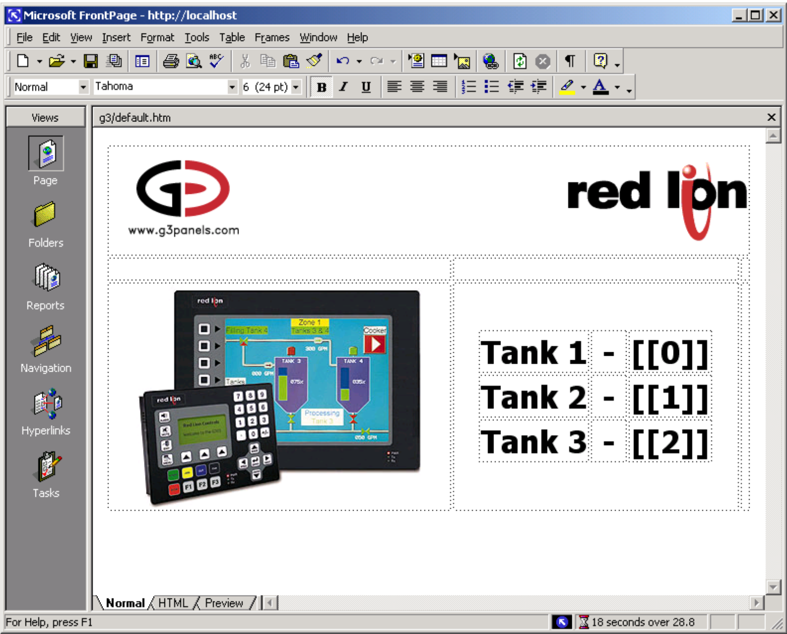


	A	B	C	D	E
1	Date	Time	Tank 1	Tank 2	Tank 3
2	01/03/03	02:28:32	50 gal	400 gal	75 gal
3	01/03/03	02:28:33	50 gal	400 gal	75 gal
4	01/03/03	02:28:34	50 gal	400 gal	75 gal
5	01/03/03	02:28:35	50 gal	400 gal	75 gal
6	01/03/03	02:28:36	50 gal	400 gal	75 gal
7	01/03/03	02:28:37	50 gal	400 gal	75 gal
8	01/03/03	02:28:38	50 gal	400 gal	75 gal
9	01/03/03	02:28:39	50 gal	400 gal	75 gal
10	01/03/03	02:28:40	50 gal	400 gal	75 gal
11	01/03/03	02:28:41	50 gal	400 gal	75 gal
12	01/03/03	02:28:42	50 gal	400 gal	75 gal
13	01/03/03	02:28:43	50 gal	400 gal	75 gal
14	01/03/03	02:28:44	50 gal	400 gal	75 gal
15	01/03/03	02:28:45	50 gal	400 gal	75 gal
16	01/03/03	02:28:46	50 gal	400 gal	75 gal
17	01/03/03	02:28:47	50 gal	400 gal	75 gal
18	01/03/03	02:28:48	50 gal	400 gal	75 gal
19	01/03/03	02:28:49	50 gal	400 gal	75 gal
20	01/03/03	02:28:50	50 gal	400 gal	75 gal
21	01/03/03	02:28:51	50 gal	400 gal	75 gal
22	01/03/03	02:28:52	50 gal	400 gal	75 gal
23	01/03/03	02:28:53	50 gal	400 gal	75 gal
24	01/03/03	02:28:54	50 gal	400 gal	75 gal
25	01/03/03	02:28:55	50 gal	400 gal	75 gal

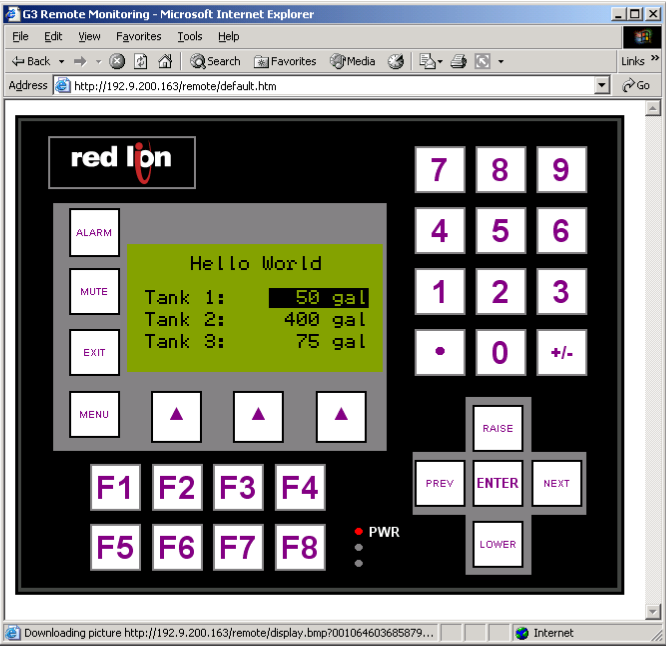
下面的图片显示的是包含三个标签的自定义页面：



下面的图片显示的是用 FrontPage 创建的自定义页面：

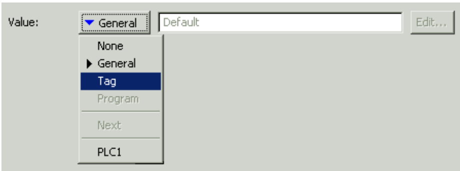


下面的图片显示的是远程浏览和控制显示：



书写表达式

在手册的前面章节中我们介绍到 Crimson 软件中很多栏目中都可以设定表达式，这些栏目是使用与下面类似的用户接口单元来设置的：



在许多情况下，你可以设定这些属性为标签值，或是远端通讯设备中的寄存器值。这种情况下，只需要简单的点击下拉菜单中的相应选项，并且从弹出对话框中选取相应数据项即可。

在某些情况下，你需要设置更为复杂的数据项集合，也许是数学运算的组合或是逻辑比较的组合。这时你就需要通过表达式来处理，在下拉菜单中选择 General 模式后，就可以在属性中输入表达式。

数据值：

表达式最少包含一个数据值，最简单的表达式可以是一个常量，一个标签或一个 PLC 寄存器。如果选择后两种情况，Crimson 为简化编辑过程，会将属性更改为相应的模式。例如：在 General 模式下输入一个标签，Crimson 将转向 Tag 模式，并在栏目中显示标签名。

常量：

常量代表一个常数或字符串。

整数型常量：

整数型常量代表一个带符号的 32 位数。它可以是十进制，二进制，八进制或十六进制。下面的例子表示一个数在四中不同进制中的表达：

进制	实例
十进制	123
二进制	0b1111011
八进制	0173
十六进制	0x7B

早期软件所支持的“U”和“L”后缀，在这里不适用。

字符常量：

字符常量代表一个 ASCII 字符，是用 32 位数的低 8 位编码的。一个字符常量可以用单引号内的一个字符表示。如'A'可表示数值 65。其它一些无法打印和表达的字符是用换码顺序来编码的，每一个都由一个反斜线引导：

顺序	数值	**ASCII**
\a	十六进制 0x07，十进制 7	BEL
\t	十六进制 0x09，十进制 9	TAB
\n	十六进制 0x0A，十进制 10	LF
\f	十六进制 0x0C，十进制 12	FF
\r	十六进制 0x0D，十进制 13	CR
\e	十六进制 0x1B，十进制 27	ESC
\xnnn	nnn 表示十六进制数	-
\nnn	nnn 表示八进制数	-
\\	一个反斜线字符	-
\'	一个单引号字符	-
\"	一个双引号字符	-

逻辑常量：

逻辑常量是一个 1 或 0 值，用来在是非判断表达式中指示真/伪。一个代表 PLC 逻辑输出的标签就可以赋予逻辑常量。逻辑常量可以以 1 或 0 表示，也可以用 true 或 false 表示。

浮点常数：

浮点常数代表一个 32 位单精度浮点数值。它是由整数部分，一个小数点和小数部分组成。指数形式不适用。

字符串常数：

字符串常数代表一组顺序字符。它是以双引号引起的一组字符。如“ABCD”是有四个字符的字符串，编译后的值是 65，66，67 和 68。（实际上，有 5 个字节存储这个字符串，在字符串后端会附加一个空字符。）换码顺序字符同样可用在字符串中。

标签值：

标签的数值是以标签名称表示的，名称不分大小写。当表达式输入后，对标签名称的任何修改，都会自动的修改所有引用该标签的表达式，所以不需要在表达式中更改。

通讯映射：

对主通讯设备中寄存器的映射可以直接在表达式中输入，其句法是方括号，寄存器名，再加方括号。寄存名前面可以加设备名作为前缀，中间以句点分隔。不加注设备名的寄存器，是指数据库中的第一个设备。下面是一些实例：

例子	意义
[D100]	第一个设备中的 D100 寄存器
[AB.N7:0]	AB 设备中的 N7：0 寄存器
[FX.D100]	FX 设备中的 D100 寄存器

简单算数：

上面提到，表达式包含多个数值，数值间可以以数学运算的形式联接。最简单的表达式可以是两个数值相加，稍复杂的表达式可以是三个数值的平均值。这些操作都使用大家都熟悉的基本句法。下面是一些基本操作的实例：

运算符	优先级	例子
加	组 4	Tag1+Tag2
减	组 4	Tag1-Tag2
乘	组 3	Tag1*tag2
除	组 3	Tag1/Tag2
求余	组 3	Tag1%Tag2

上面的例子中，操作数前后加了空格，实际中是不需要的。

操作数的优先级：

你已经注意到上表中的优先级栏。从我们的代数知识中可知，当多个操作数一齐使用时，它们是按设定的顺序进行运算的。例如：乘法在加法之前运算。Crimson 使用操作数的优先级形式完成这种顺序，操作数按从低到高的顺序依此进行运算（除上面提到的外，同一优先级的操作数按从左至右顺序进行）。当使用括号后原定的顺序将会改变。

类型转换：

通常，Crimson 会自动判别何时将整数型的表达式转换为浮点型的表达式进行运算。例如，如果将一个整数除以一个浮点数值，整数在运算前将转换为浮点数。然而在很多情况下，你可以强制执行这种转换。

例如：现要将 3 个代表油箱液位的整数相加，再除以油箱数得到液位平均数。如果使用表达式(Tank1+Tank2+tank3)/3，那么结果可能不如你需要的那么精确。除法使用整数运算，平均值不包括任何小数。要使 Crimson 使用浮点运算，最简单的方法是将 3 改为 3.0，Crimson 在除法运算前会将整数转换为浮点数。稍复杂一点的方法是使用句法 float(Tank1+Tank2+tank3)/3。这是使用术语称“类型指定”的方法，手动将整数转换为浮点数。

类型指定方法也可以将浮点数转换为整数，可能在存储至 PLC 寄存器前将过程值降低精度。例如：表达式 int(cos(Theta)*100)使用浮点运算计算一个角度的余弦值乘以 100，然后省略小数位转换为整数值。

比较数值：

你会经常需要比较两个数值，根据结果做出判断。例如：你要定义一个标志型公式来表示油箱液位是否达到设定值，或者用 if 语句在电机达到预定转速时执行一些代码。下面是一些比较运算符：

运算符	优先级	例子
等于	组 7	Data = = 100
不等于	组 7	Data ! = 100
大于	组 6	Data > 100
大于等于	组 6	Data > = 100
小于	组 6	Data < 100
小于等于	组 6	Data <= 100

每个运算符按条件的不同，会产生一个 1 或 0 的结果。运算符可以对整数，浮点数和字符串进行处理。如果是字符串比较，将不分大小写。“abc” 等于“ABC”。

测试位：

Crimson 允许你使用位选择运算符对一个数据的位进行测试，运算符是一个句点。运算符的左手是要测试的数据，右手可以是一个表达式代表要测试的位，它的值介于 0~31 间。按位的值的不同，运算结果等于 1 或 0。

运算符	优先级	例子
位选择	组 1	Input.2

上面的例子是在指定的标签中测试第二位（该位的值为 1）。

如果你要测试位值等于 0，你可用逻辑非运算符：

运算符	优先级	例子
逻辑非	组 2	！ Input.2

如果 2 位的值等于 0，则上面的例子等于 1，反之亦然。

多重条件：

如果在多个条件都为真时，设定表达式为真，可以使用逻辑与运算；如果在任一条件为真时，表达式为真，可使用逻辑或运算。下面是两个运算的例子：

运算符	优先级	例子
逻辑与	组 11	A>10 && B>10
逻辑或	组 12	A>10 || B>10

当运算符两边的表达式都为真时，与运算产生数值 1；当运算符两边的表达式任一个为真时，或运算产生数值 1。注意：逻辑运算在答案已经得出后，立即停止执行运算。如在上面的与运算例子中，运算符右边的比较只有在 A 大于己于 10 时才进行，如果 A 小于 10 则结果已经等于 0。但当左边或右边的表达式是调用一个程序或改变一个数据值时，则上面的属性不适用。

挑选数值：

在有些情况下，你需要按条件在两个数据间选择，数据可以是整数型，浮点型或字符串型。例如：按一个标志型标签的状态，设定电机的转速为 500rpm 或 2000rpm，这可用运算符？：来执行，它有 3 个变量，看示例：

运算符	优先级	例子
选择	组 13	Fast ? 2000 : 500

当 Fast 为真时，选择 2000，反之选 500。运算符与 if 语句的功能相同。

位操作：

Crimson 提供一些运算符对整数按位进行处理。这些运算符称为位运算符。

与，或和异或：

这三个位运算符按左右两边数值的不同产生不同结果：

运算符	优先级	例子
位与	组 8	Data & Mask
位或	组 9	Data | Mask
位异或	组 10	Data ^ Mask

下面是三种运算的真值表：

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

移位运算符：

Crimson 提供对整数进行左右移位的运算符：

运算符	优先级	例子
左移	组 5	Data << 2
右移	组 5	Data >> 2

上面例子中对 Data 按方向移动 2 位。

位非：

最后 Crimson 还提供对数值按位取反的位非运算符：

运算符	优先级	例子
位非	组 2	~Mask

上面的例子中，Mask 中的每一位将取反，得到一个数值。

数组的索引：

数组内的每一个数据单元都可以通过数组名和包含索引表达式的方括号来引用。索引表达式的值在 0 至单元数-1 的范围内。例如：一个数组有 10 个单元，第一个单元是 Name[0]，最后一个单元是 Name[9]。

字符串索引：

方括号同样可以用来在一个字符串中选取字符。如：有一个名为 Text 的标签包含字符串“ABCD”，表达式 Text[0]将返回数值 65，这是第一个字符的 ASCII 码值。索引号超出字符串范围会返回 0 值。

字符串相加：

与数字相加一样，加法运算符也可以用于字符串的连接。“AB”+“CD”等于“ABCD”。也可以将整数与字符串相加，这时整数代表单个字符的 ASCII 码值，会连接到字符串中。

调用程序：

程序的返回值可以用程序名和一对括号来引用，并在表达式中使用。例如：Program1()*10 就是调用一个程序的返回值，并乘以 10。当然 Progarm1 必须返回一个整数或浮点数。

使用函数：

Crimson 提供许多预定的函数来访问系统信息，或进行数学运算。这些函数在函数索引这一章节详细介绍。调用函数的句法与调用程序的非常类似。功能中包含的所有变量都以括号围绕。例如：cos(0) 是调用余弦函数，变量值是 0，返回值是+1.0。

优先级总结：

<table>
<tr><th>组</th><th>运算符</th></tr>
<tr><td>组 1</td><td>.</td></tr>
<tr><td>组 2</td><td>! ~</td></tr>
<tr><td>组 3</td><td>* / %</td></tr>
<tr><td>组 4</td><td>+ -</td></tr>
<tr><td>组 5</td><td><< >></td></tr>
<tr><td>组 6</td><td>< > <= >=</td></tr>
<tr><td>组 7</td><td>== !=</td></tr>
<tr><td>组 8</td><td>&</td></tr>
<tr><td>组 9</td><td>|</td></tr>
<tr><td>组 10</td><td>^</td></tr>
<tr><td>组 11</td><td>&&</td></tr>
<tr><td>组 12</td><td>||</td></tr>
<tr><td>组 13</td><td>?:</td></tr>
</table>

数值越低的组优先操作。

Edict 用户须知

Edict97 用户必须注意：

- &&和|| 运算在结果确定后立刻停止操作。
- 数据类型只支持字符串型，整数型和浮点型。
- ？：支持 3 重操作。

编写动作

表达式用来设定数值，而动作是在触发器和事件发生时所要执行的操作。由于大多数的动作是与按键相联系的，同时 Crimson 用标准对话框的形式来定义最常用的动作，你可以避免人工编写这些动作。当你需要使用触发器，写程序或用户定义模式的按键时，你才需要编写动作。

更换页面 ：

要创建一个动作来更改当前显示的页面时，使用句法 GotoPage(Name)，这里 Name 指要显示的页面名。当前页面被移除，新页面显示在屏幕上。

更改数值：

Crimson 有多种方法改变数据值。

简单赋值：

要创建一个动作来对一个标签或通讯设备中的寄存器赋予新值，使用句法 Data:=Value，这里 Data 指要更改的数据项，Value 指要赋予的数值。注意：Value 不仅仅是常数，还可以是任何正确类型的有效表达式。可参考前面的章节中如何书写表达式。例如：[N7:0]:=Tank1+Tank2 是将两个油箱的液位值相加，将总和存储在 PLC 寄存器中。

复合赋值：

要创建一个动作，来设置数值等于当前值与其它数值经各种运算后的结果，使用句法 Data op=Value，这里 Data 是要更改的标签，Value 是用于运算的数值，op 是任意可用的运算符。例如：Tag+=10 表示 Tag 加 10，Tag*=10 表示 Tag 乘 10。

递增和递减：

要创建一个动作，对一个数据值加 1,可使用句法 Data++。要创建一个动作，对一个数据值减 1,可使用句法 Data--。运算符++和--可按需要放在数据值的前面或后面。放在前面时，表达式++Data 的值等于数值加 1 后的值；放在后面时，表达式的值等于数值加 1 前的值。

更改位值：

改变标签内一个位的值，可用句法 Data.Bit:=1 或 Data.Bit:=0 置位或复位。Data 是操作的标签，Bit 是 0 起点的位数。注意：运算符右边可以是表达式，如：Data.1：=（Level>10），按液位是否超过设定值，进行置位或复位。

运行程序：

程序可以用程序名带括号的方式在动作中调用。如：Program1()。按设定的程序属性，程序可在前台或后台执行。

使用函数：

Crimson 提供很多预定义的函数，可完成多种功能。这些函数在函数索引章节中有详细的介绍。函数的引用句法与程序很类似，只是在括号中包含了变量，如：Setlanguage(1)，设置屏的语言为语言 1。

运算符优先级：

所有的赋值运算符列为第 14 组，也就是说，它们是最后执行的运算符。在同组中它们是从右至左执行的，Tag1:=Tag2:=Tag3:=0，会同时对 3 个标签复位。

Edict 用户须知

Edict97 用户必须注意：

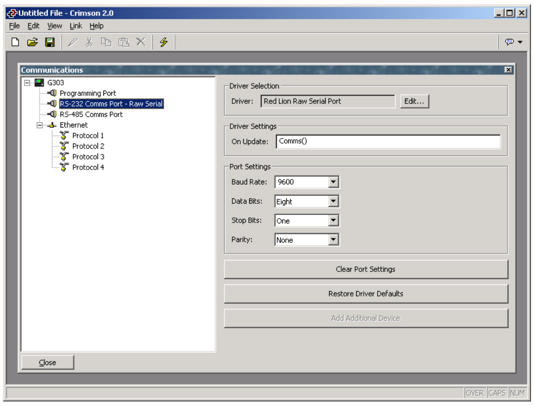
- =运算符现在代替 := 运算符。

使用原始通讯口

为了使用户可以编写自己的简单 ASCII 协议，Crimson 提供了一个新的功能，使用编程语言直接控制串行通讯口或 TCP/IP 网口。这个功能称为原始通讯口访问，代替了 Edict97 中的 Roll-Your-Own Protocol 功能。它也代替了通用 ASCII 帧协议中的解析操作。注意：如果你只使用 Crimson 提供的标准协议，而没有使用客户自定义协议，可以跳过这节。

设置串行通讯口：

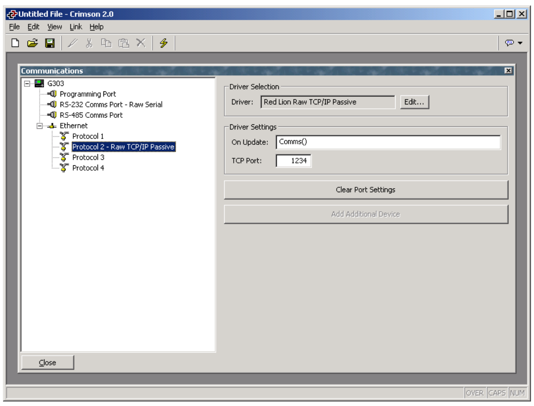
要在原始模式下使用串行通讯口，选择 Raw Serial Port 协议：



通讯口的波特率和字节格式应按需要的通讯设置，On Update 属性设置执行通讯的程序。这个程序会在通讯口的通讯任务中持续调用。

设置 TCP/IP 口：

要在原始模式下使用 TCP/IP 口，选择 Raw Raw TCP/IP Passive 协议：



On Update 属性的设定与上面相同，Port 属性设置你需要协议监控哪一个 TCP 口。协议将接受该端口的连接，并使用 On Update 中的程序处理通讯。

读字符：

使用 PortRead 函数可一次从串口读取一个字符，详细内容见函数索引章节。在所有的原始通讯口函数中，port 变量是按在通讯窗口的左手窗中通讯口列表来计数的，编程口是端口 1。

下面是使用 PortRead 读取字符的例子：

```
int Data;
for (; ;) {
        If ( ( Data := PortRead (2, 100) ) >= 0 ) {
                /* Add code to process data */
                }
        }
```

注意：如果在 period 变量中输入非零值，就不必调用 Sleep 函数了。如果设置 period 变量为零，你必须知道有时会暂停通讯任务，会影响系统操作。

读取一帧：

使用 PortInput 函数可一次从串口读取一个数据帧，详细内容见函数索引章节。该函数可以设置帧分隔符，帧长度和帧输入时间。所以不需要编写自己的接收器。下面是实例：

```
cstring input;
int     value;

for ( ; ; )     {

        input := PortInput (5, 42, 13, 0, 0) ;

        if ( value := TextToInt (input, 10) ) {

                Speed := value;

                PortPrint (5, “Value is ”);
                PortPrint (5, IntToText(value,10,5));
                PortPrint (5, “\r\n ”);
                }
        }
```

上面的例子是从 TCP/IP 口接受一个数据帧，以星好起始，以回车结束。将数据帧转换为十进制数，存储在整数寄存器中，将数值应答返回客户端。

发送数据：

使用 PortWrite 函数和 PortPrint 函数通过串口发送数据，详细内容见参考函数章节。第一个函数发送单个字节，第二个函数发送整个字符串。要发送数值使用 IntToText 函数进行转换。

Edict 用户须知

Edict97 用户必须注意：

- Raw serial port device port device 协议控制通讯口的握手信号线，所以 SetRTS，HoldTx 和类似通讯口管理的函数都不再使用，Crimson 也不提供这些函数。
- 在发送数据时，Crimson 自动处理缓冲器溢出事件，以保证数据不丢失。所以 PortWrite 和 PortPrint 函数不提供返回值。
- 要直接的模拟 GAF，可使用与上面的例子类似的程序，只是将接收字符串存储在字符串标签中，同时对整数标签增量。但这不是好的方法，因为需要用触发器来响应序列数的改变，你可以在通讯程序中处理这个逻辑。
- Crimson 在使用原始通讯口时可使用高级编程，它的性能会超过 Edict 很多。

系统变量引用：

下面的几页将描述 Crimson 中系统变量的使用，在前面的两章中我们已经介绍了怎样在动作和表达式中调用系统变量。

系统变量的用途：

系统变量用来反映系统状态，或更改系统特性。前一种变量是只读的，后一种变量是可写的。

DispUpdates

变量	类型	说明
value	int	显示更新速度的数值

说明:

返回数值指示显示更新速度

变量类型:

整数

访问类型:

只读

DispContrast

变量	类型	说明
value	int	显示对比度的百分比数值

说明:

返回数值指示显示对比度，从 0 到 100 的百分比，为 0 时无显示图形。

变量类型：

整数

访问类型：

读/写

DispBrightness

变量	类型	说明
value	int	显示亮度的百分比数值

说明:

返回数值指示显示对比亮度，从 0 到 100 的百分比，为 0 时无显示。

变量类型：

整数

访问类型：

读/写

PI

变量	类型	说明
value	int	数值 3.14159274

说明:

返回浮点数 pi。

变量类型：

浮点数

访问类型：

只读

函数索引

下面将介绍 Crimson 中各种函数，这些函数可以在各种动作和表达式中引用。在函数中，主动函数不能用于不可以改变表达式的场合，如：控制元件显示的表达式；而被动函数可以用于任何场合。

Edict 用户须知：

- Edict97 中 Serial RYOP 函数已经被多个 port 函数代替。

ABS(*Value*)

变量	类型	说明
value	int / float	需计算的数值

说明:

对变量取绝对值后返回数值。如果 Value 是正数，直接将数值返回；如果 Value 是负值，将负号变为正号后将数值返回。

函数类型：

被动函数

返回值类型：

整数或浮点数。随 Value 变量的类型而定。

实例：

Error := abs(PV – SP)

ACOS(*Value*)

变量	类型	说明
value	Float	需计算的数值

说明：

反余弦函数，返回值是角度，弧度单位。

函数类型：

被动函数

返回值类型：

浮点数

实例：

theta := acos(1.0)

ASIN(*Value*)

变量	类型	说明
value	float	需计算的数值

说明：

反正弦函数，返回值是角度，弧度为单位

函数类型：

被动函数

返回值类型：

浮点数

实例：

```
theta := asin(1.0)
```

ATAN(*Value*)

变量	类型	说明
value	float	需计算的数值

说明：

反正切函数，返回值是角度，弧度为单位。

函数类型：

被动函数

返回值类型：

浮点数

实例：

theta := atan(1.0)

ATAN2(*A, B*)

变量	类型	说明
a	float	角的对边值
b	float	角的邻边值

说明

该函数等于 atan(a/b)， a 和 b 的符号可确定相应的象限。当 b 等于 0 时，函数有效，可处理单变量函数 tan 产生的无限值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

theta := atan2(1,1)

BEEP(*Freq, Period*)

变量	类型	说明
freq	int	半音频率设定
period	int	鸣响时间 ， 毫秒为单位

说明：

界面中的蜂鸣器按设定时间和间隔鸣响。Period 等于 0，将关闭蜂鸣器。Beep 函数不按队列顺序执行， 新的调用会立即更改以前的设定。改变变量 Freq 可改变音色。Beep 函数在提示不同的事件和程序执行过程中非常实用。

函数类型：

主动函数

返回值类型：

无返回值

实例：

Beep(60, 100)

ClearEvents()

变量	类型	说明
无		

说明：

清除事件记录器中显示的事件列表

函数类型：

主动函数

返回值类型：

无返回值

实例：

ClearEvents()

CloseFile(*File*)

变量	类型	说明
file	int	对用 OpenFile 函数返回的待处理文件.

说明：

关闭用 OpenFile（ ）函数打开的文件。

函数类型：

主动函数

返回值类型：

无返回值

实例：

CloseFile(hFile)

CompactFlashEject()

变量	类型	说明
无		

说明：

停止对 CF 卡的访问，以安全的拔出 CF 卡。

函数类型：

主动函数

返回值类型：

无返回值

实例：

CompactFlashEject()

CompactFlashStatus()

变量	类型	说明
无		

说明：

返回一个整数值，指示 CF 卡槽的状态。

数值	状态	说明
0	空	无 CF 卡或已调用 **CompactFlashEject** 函数将卡弹出。
1	无效	卡损坏，无格式化或格式化不正确。 注意：仅支持 FAT16 格式。
2	校验	G3 正在校验卡的状态。仅在卡第一次插入 G3 时出现。
3	格式化	G3 正在对卡进行格式化，此状态在 PC 机发出格式化命令时出现。
4	已锁定	G3 正在对卡进行写操作，或 Windows 正在访问卡。
5	已安装	卡安装完毕，并且未被 G3 或 Windows 锁定。

函数类型：

主动函数

返回值类型：

整数型

实例：

d := CompactFlashStatus()

ControlDevice(*Device, Enable*)

变量	类型	说明
device	int	被使能或禁用的设备
enable	int	设定使能或禁用

说明：

允许数据库文件使能或禁用指定的通讯设备。变量 device 是设备索引号，当在通讯窗口中选择了设备名后，设备索引号会显示在状态栏中。

函数类型：

主动函数

返回值类型：

无返回值

实例：

ControlDevice(1, true)

Copy(*Dest, Src, Count*)

变量	类型	说明
dest	int / float	目的地址的第一个数组单元
src	int / float	源地址的第一个数组单元
count	int	单元个数

说明：

从 src 地址向 dest 地址复制 count 个单元的数组数据。

函数类型：

主动函数

返回值类型：

无返回值

实例：

Copy(Save[0], Work[0], 100)

COS *(Theta)*

变量	类型	说明
theta	float	计算的角度，弧度为单位

说明:

返回角度 theta 的余弦函数值。

函数类型：

被动函数

返回值类型:

浮点数

实例：

xp := radius*cos(theta)

CreateDirectory(*Name*)

变量	类型	说明
name	cstring	需创建的目录名

说明：

在 CF 卡上创建新的目录。如有错误，返回值为 0

函数类型：

主动函数。

返回值类型：

整数

实例：

Result := CreateDirectory(“/LOGS/LOG1”)

CreateFile(*Name*)

变量	类型	说明
name	cstring	需创建的文件名

说明：

在 CF 卡上创建一个文件。如有错误，返回值为 0

函数类型：

主动函数。

返回值类型：

整数

实例：

Success := CreateFile(“/logs/custom/myfile.txt”)

DataToText(*Data, Limit*)

变量	类型	说明
data	int	数组中的第一个单元
limit	int	单元个数

说明：

按数组 data 中的数据建立一个字符串，每个数组单元代表一个 ASCII 字符。

函数类型：

被动函数。

返回值类型：

字符串

实例：

string := DataToText(Data[0], 8)

Date(*Y, M, D*)

变量	类型	说明
y	int	年份，4 位数形式
m	int	月份，从 1 到 12.
d	int	日期，从 1 开始

说明：

返回一个数值，它代表从 1997 年 1 月 1 日起到指定日期的总秒数时间。此数值用于其它日期和时间函数。

函数类型：

被动函数。

返回值类型：

整数

实例：

t := Date(2000,12,31)

DecToText(*Data, Signed, Before, After, Leading, Group*)

变量	类型	说明
data	int/float	需格式转换的数值
signed	int	0 – 无符号, 1 – 软符号, 2 – 硬符号.
before	int	小数点左边的数据位数
after	int	小数点有边的数据位数
leading	int	0 – 带起始 0, 1 –不带起始 0
group	int	0 – 不分组, 1– 3 位数 1 组

说明：

按相应的格式设定将 data 中的十进制数转换为 ASCII 码的数值。此函数通常是在程序中调用，并且为串行通讯口的数据发送作准备。

函数类型：

被动函数

返回值类型：

字符串

实例：

Text := DecToText(var1, 2, 5, 2, 1, 1)

Deg2Rad(*Theta*)

变量	类型	说明
theta	float	需计算的角度.

说明：

返回将 theta 由角度转换为弧度后的数值。

函数类型：

被动函数。

返回值类型：

浮点数

实例：

```
Load := Weight * cos(Deg2Rad(Angle))
```

DeleteDirectory(*Name*)

变量	类型	说明
name	cstring	需删除的目录名

说明：

从 CF 卡中删除一个目录，及其子目录和内容。注意：文件系统不支持长文件名；如果用反斜线号来间隔文件路径，按 Crimson 的规则，须按字符串处理，文件名和路径用双引号引起，详见书写表达式章节。如果用斜线号代替反斜线号，则不必使用双引号。此函数执行完毕则返回 1，执行失败返回 0。

函数类型：

主动函数

返回值类型：

整数

实例：

Success := DeleteDirectory("/logs/custom")

DeleteFile(*File*)

变量	类型	说明
file	int	由 OpenFile 函数返回的待处理文件

说明：

关闭并删除 CF 卡上的指定文件。

函数类型：

主动函数

返回值类型：

整数型

实例：

Result := DeleteFile(hFile)

DevCtrl(*Device, Function, Data*)

变量	类型	说明
device	int	需控制的设备索引号
function	int	需执行的功能.
data	cstring	功能的附加参数

说明：

此函数是对通讯设备执行一个特殊操作。变量 device 指定相应设备索引号，当设备被选择后，状态栏中会显示设备索引号。参数 function 设置所需的操作，它的值按设备类型而定。参数 data 是向设备驱动器发送附加参数。多数驱动不支持此参数，当有此参数时，按驱动的规定来设置，会有单独说明。

函数类型：

主动函数

返回值类型：

整数型

实例：

实例可参考通讯驱动单独应用说明。

DisableDevice(*Device*)

变量	类型	说明
device	int	待禁用的设备

说明：

禁用与指定设备的通讯。变量 device 是设备索引号，在通讯窗口中选定设备后，设备索引号会显示在状态栏中。

函数类型：

被动函数

返回值类型：

无返回值

实例：

DisableDevice(1)

DispOff()

变量	类型	说明
none	float	关闭背光灯

说明：

关闭背光灯。

函数类型：

主动函数。

返回值类型：

无返回值

实例：

DispOff()

DispOn()

变量	类型	说明
none		打开背光灯

说明：

打开背光灯。

函数类型：

主动函数。

返回值类型：

无返回值

实例：

DispOn()

DrvCtrl(*Port, Function, Data or Value???*)

变量	类型	说明
port	Int	驱动所控制的端口索引号
function	Int	需执行的功能
data	Cstring	该功能包含的参数

说明：

此函数是对通讯驱动执行一个特殊操作。变量 port 指定通讯驱动所控制的端口号。变量 function 设置所执行的操作，它的值由驱动本身来规定。变量 data 是向驱动传送的附加参数。多数驱动不支持此功能，当有此功能时，按驱动的规定设来置，会有单独说明。

函数类型：

主动函数

返回值类型：

整数型

实例：

实例可参考通讯驱动单独应用说明。

EnableDevice(*Device*)

变量	类型	说明
device	int	待使能的设备

说明：

使能与指定设备的通讯。变量 device 是设备索引号，在通讯窗口中选定设备后，设备索引号会显示在状态栏中。

函数类型：

被动函数

返回值类型：

无返回值

实例：

EnableDevice(1)

Exp(*Value*)

变量	类型	说明
value	float	需计算的数值

说明：

计算 e (2.7183)的 value 次幂。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Variable2 := exp(1.609)

Exp10(*Value*)

变量	类型	说明
value	float	需计算的数值

说明：

计算 10 的 value 次幂。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Variable4 := exp10(0.699)

Fill(*Element, Data, Count*)

变量	类型	说明
element	int / float	待处理的第一个数组单元
data	int / float	写入的数值
count	int	单元个数

说明：

将 data 写入数组单元 element 起始的 count 个单元。

函数类型：

主动函数

返回值类型：

无返回值。

实例：

Fill(List[0], 0, 100)

Find(*String, Char, Skip*)

变量	类型	说明
string	cstring	需处理的字符串
char	int	需查询的字符
skip	int	跳过字符的次数

说明：

返回 string 字符串中 char 字符的位置值，并跳过 skip 个 char 字符。第一个字符的位置是 0；如没找到 char 字符返回-1。在下面的例子中，跳过第一个“：”后，“：”的位置是 7。

函数类型：

被动函数

返回值：

整数

实例：

Position := Find("one:two:three",':',1)

FindFileFirst(*Dir*)

变量	类型	说明
dir	cstring	Directory to be used in search.

说明：

返回 CF 卡中 dir 目录中的第一个文件或目录名字。如果没有文件或无 CF 卡，则返回空字符串。该函数与 FindFileNext 函数一起用来检索指定目录中的文件。

函数类型：

主动函数

返回值：

字符串

实例：

Name := FindFileFirst(“/LOGS/LOG1”)

FindFileNext()

变量	类型	说明
none		

说明：

在调用 FindFileFirst 函数后，该函数返回下一个文件或目录的名字。如果没有文件，则返回空字符串。该函数与 FindFileFirst 函数一起用来检索指定目录中的文件。

函数类型：

主动函数

返回值：

字符串

实例：

Name := FindFileNext()

FormatCompactFlash()

变量	类型	说明
none		

说明：

格式化人机界面中的 CF 卡，会删除卡上的所有数据。在调用该函数前，应给操作者必要的告警信息。

函数类型：

主动函数

返回值类型：

无返回值

实例：

FormatCompactFlash()

GetDate(*Time*) and Family

变量	类型	说明
time	int	经编码的时间值

说明：

这一系列的函数可根据 GetNow，Time 或 Date 等函数的返回值来计算出相应的时间/日期值。可用的函数如下：

函数	说明
GetDate	在 time 值中返回日期值。
GetDay	在 time 值中返回星期值。
GetDays	在 time 值中返回天数值。
GetHour	在 time 值中返回小时值。
GetMin	在 time 值中返回分钟值。
GetMonth	在 time 值中返回月份值。
GetSec	在 time 值中返回秒钟值。
GetWeek	在 time 值中返回年计的星期数值。
GetWeeks	在 time 值中返回周数值。
GetWeekYear	在使用星期数时返回年份值。
GetYear	在 time 值中返回年份值。

注意：

GetDays 和 GetWeeks 函数用来计算两个时间差值的天数和周数。GetWeekYear 和 GetYear 不同，当在一年的最后一周跨年时，前者中返回值小于后者。

函数类型：

被动函数

返回值：

整数

实例：

d := GetDate(GetNow() – 12*60*60)

GetInterfaceStatus(*Interface*)

变量	类型	说明
interface	int	查询的接口

说明：

返回一个字符串来指示选定 TCP/IP 接口的状态。参考前面章节中关于高级通讯的介绍，就可以了解怎样计算 interface 中的数值，以及返回值的含义。

函数类型：

被动函数

返回值：

字符串

实例：

EthernentStatus := GetInterfaceStatus(1)

GetMonthDays(*Y, M*)

类型	Type	说明
y	int	年份，4 位数格式.
m	int	月份，1 至 12

说明：

返回指定年月的天数，包括闰年。

函数类型：

被动函数

返回值类型：

整数

实例：

Days := GetMonthDays(2000, 3)

GetNetGate(*Port*)

变量	类型	说明
port	int	以太网口的索引号，必需设为 0

说明：

返回缺省网关的 IP 地址，以点-数字间隔文本形式。

函数类型：

被动函数

返回值类型：

字符串

实例：

gate := GetNetGate(0)

GetNetID(*Port*)

变量	类型	说明
port	int	以太网口的端口号，必须是 0

说明：

返回以太网口的 MAC 地址，以 17 个字符文本形式。

函数类型：

被动函数

返回值类型：

字符串

实例：

MAC := GetNetId(0)

GetNetIP(*Port*)

变量	类型	说明
port	int	以太网口的端口号，必须是 0

说明：

返回以太网口的 IP 地址，以点-数字间隔文本形式。

函数类型：

被动函数

返回值类型：

字符串

实例：

IP := GetNetIp(0)

GetNetMask(*Port*)

变量	类型	说明
port	int	以太网口的端口号，必须是 0

说明：

返回以太网口的 IP 地址掩码，以点-数字间隔文本形式。

函数类型：

被动函数

返回值类型：

字符串

实例：

mask := GetNetMask(0)

GetNow()

变量	类型	说明
none		

说明：

返回从 1997 年 1 月 1 日至当前时刻所经过的总秒数时间。该值用于其它时间/日期函数。

函数类型：

被动函数

返回值类型：

整数

实例：

t := GetNow()

GetNowDate()

变量	类型	说明
无		

说明：

返回从 1997 年 1 月 1 日至当前日期所经过的总秒数时间。

函数类型：

被动函数

返回值类型：

整数

实例：

d: = GetNowDate()

GetNowTime()

变量	类型	说明
无		

说明：

以总秒数时间的形式返回当前时间。

函数类型：

被动函数

返回值类型：

整数

实例：

t := GetNowTime()

GetUpDownData(*Data, Limit*)

变量	类型	说明
data	int	一个恒定增加数据
limit	int	产生数据的个数

说明：

此函数可将一个恒定增加的数值，转换为 0 到 limit-1 间返回值。通常它是用在演示程序中产生一个动画图形。通常将系统变量 DispCount 设为 data，在每次页面更新时结果数值都会改变。如果函数 GetUpDownStep 使用同样变量，它的返回值可指示由 GetUpDownData 返回的数据的变化方向。

函数类型：

被动函数

返回值类型：

整数

实例：

Data := GetUpDownData(DispCount, 100)

GetUpDownStep(*Data, Limit*)

变量	类型	说明
data	int	一个恒定增加数据
limit	int	产生数据的个数

说明：

见 GetUpDownData 中的说明

函数类型：

被动函数

返回值类型：

整数

实例：

Delta := GetUpDownStep(DispCount, 100)

GotoPage(*Name*)

变量	类型	说明
name	Display Page	显示的页面名

说明：

调入要显示在屏幕上的页面。

函数类型：

主动函数

返回值类型：

无返回值.

实例：

GotoPage(Page1)

GotoPrevious()

变量	类型	说明
无		

说明：

调入上次显示的页面。

函数类型：

主动函数

返回值类型：

无返回值.

实例：

GotoPrevious()

HidePopup()

变量	类型	说明
无		

说明：

退出用 ShowPopup 函数调出的页面

函数类型：

主动函数

返回值类型：

无返回值.

实例：

HidePopup()

IntToText(*Data, Radix, Count*)

变量	类型	说明
data	int	处理的数据
radix	int	使用的进制
count	int	产生的位数

说明：

按规定的进制和位数将一个数值转换为代表数值的字符串。radix 代表进制，count 代表位数。数值默认为无符号数，如果需要有符号数，先用 Sgn 判断是否加负号前缀，再用 Abs 对数值取绝对值后，用 IntToText 进行转换。

函数类型：

被动函数

返回值类型：

字符串

实例：

PortPrint(1, IntToText(Value, 10, 4))

IsDeviceOnline(*Device*)

变量	类型	说明
device	int	需反馈设备的端口号

说明：

反馈指定设备是否在线。当出现连续通讯错误，判定设备离线。当设备处于离线状态时，会周期性的对设备轮询，以确定设备是否返回在线状态。

函数类型：

被动函数

返回值类型：

整数

实例：

Okay := IsDeviceOnline(1)

Left(*String, Count*)

变量	类型	说明
string	cstring	需处理的字符串
count	int	返回的字符数

说明：

从字符串 string 中返回前 count 个字符

函数类型：

被动函数

返回值类型：

字符串

实例：

AreaCode := Left(Phone, 3)

Len(*String*)

变量	类型	说明
string	cstring	需处理的字符串

说明：

返回字符串 string 中字符的个数

函数类型：

被动函数

返回值类型：

整数

实例：

Size := Len(Input)

Log(*Value*)

变量	类型	说明
value	float	需处理的数值

说明：

返回变量 value 的自然对数值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Variable1 := log(5.0)

Log10(*Value*)

变量	类型 **Type**	说明
value	float	需处理的数值

说明：

返回变量 value 以 10 为底的对数值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Variable3 := log10(5.0)

MakeFloat(*Value*)

变量	类型	说明
value	int	需转换的数值

说明：

将整数型的变量转译为浮点数型。该函数并不是执行数据类型转换，而是在变量中存储一个位形式，假设它不代表整数，而是代表浮点数。通常，该函数是用来处理远端设备中的数据，以满足通讯协议的要求。

函数类型：

被动函数

返回值类型：

浮点数

实例：

fp := MakeInt(n);

MakeInt(*Value*)

变量	类型	说明
value	float	需转换的数值

说明：

将浮点数型的变量转译为整数型。该函数并不是执行数据类型转换，而是在变量中存储一个位形式，假设它不代表浮点数，而是代表整数。通常，该函数是用来处理远端设备中的数据，以满足通讯协议的要求。

函数类型：

被动函数

返回值类型：

整数

实例：

n := MakeInt(fp);

Max(*A, B*)

变量	类型	说明
a	int / float	第一个比较值
b	int / float	第二个比较值

说明：

返回两个变量中的较大值

函数类型：

被动函数

返回值类型：

整数或浮点数，按变量类型而定。

实例：

Larger := Max(Tank1, Tank2)

Mean(*Element, Count*)

变量	类型	说明
element	int / float	待处理的第一个数组单元
count	int	数组单元个数

说明：

返回指定数组单元中数据的平均值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Average := Mean(Data[0], 10)

Mid(*String, Pos, Count*)

变量	类型	说明
string	cstring	需处理的字符串
pos	int	起始位置
count	int	返回字符的个数

说明：

从字符串 string 中返回以 pos 为起始位置的第 count 个字符，第一个字符位置是 0。

函数类型：

被动函数

返回值类型：

字符串

实例：

Exchange := Mid(Phone, 3, 3)

Min(*A, B*)

变量	类型	说明
a	int / float	第一个比较值.
b	int / float	第二个比较值.

说明：

返回两个变量中的较小值

函数类型：

被动函数

返回值类型：

整数或浮点数，按变量类型而定。

实例：

Smaller := Min(Tank1, Tank2)

MulDiv(*A, B, C*)

变量	类型	说明
a	int	第一个值
b	int	第二个值
c	int	第三个值

说明：

返回 a*b/c 的结果。中间值按 64 为整数计算，以避免溢出。

函数类型：

被动函数

返回值类型：

整数。

实例：

d := MulDiv(a, b, c)

MuteSiren()

变量	类型	说明
无		

说明：

关闭人机界面中的蜂鸣器。

函数类型：

主动函数

返回值类型：

无返回值

实例：

MuteSiren()

Nop()

变量	类型	说明
无		

说明：

该函数为空操作。

函数类型：

被动函数

返回值类型：

无返回值

实例：

Nop()

OpenFile(*Name, Mode*)

变量	类型	说明
name	cstring	需打开的文件
mode	int	打开文件的模式 0 = 只读 1 = 从文件头读/写 2 = 从文件尾读/写

说明：

打开 CF 卡上的 name 文件。该函数最多可同时打开 4 个文件，当文件打开时 CF 卡不能卸载。注意：文件系统不支持长文件名；如果用反斜线号来间隔文件路径，按 Crimson 的规则，须按字符串处理，文件名和路径用双引号引起，详见书写表达式章节。如果用斜线号代替反斜线号，则不必使用双引号。该函数不能创建新文件，在调用该函数前必须用 CreatFile()函数创建文件。

函数类型：

主动函数

返回值类型：

整数

实例：

hFile := OpenFile(“/LOGS/LOG1/01010101.csv”, 0)

Pi()

变量	类型	说明
无		

说明：

返回浮点数π。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Scale = Pi()/180

PlayRTTTL(*Tune*)

变量	类型	说明
tune	cstring	RTTTL 格式的曲调

说明:

用内部蜂鸣器播放曲调。变量 tune 应包含 RTTTL 格式的曲调。该格式的曲调一般用于移动电话，作为自选铃声。下面的例子是从互联网中下载的。

函数类型：

主动函数

返回值类型：

无返回值

实例：

PlayRTTTL("TooSexy:d=4,o=5,b=40:16f,16g,16f,16g,16f.,16f,16g,16f,16g,16g#.,16g#,16g,16g#,16g,16f.,16f,16g,16f,16g,16f.,16f,16g,16f,16g,16f.,16f,16g,16f,16g,16g#.,16g#,16g,16g#,16g,16f.,16f,16g,16f,16g,32f.")

PopDev(*Element, Count*)

变量	类型	说明
element	int / float	需处理的第一个数组单元
count	int	单元个数

说明：

假设数组代表所调查总人口，此函数会返回数组中 element 单元为起始的，count 个数组单元中数据的标准偏差。如需要计算人口采样的标准偏差，使用函数 StdDev。

函数类型：

被动函数

返回值类型：

浮点值

实例：

Dev := PopDev(Data[0], 10)

PortClose(*Port*)

变量	类型	说明
port	int	要关闭的指定端口

说明：

在使用主或从原始 TCP 端口协议时，该函数用关闭相应插座连线的方法来关闭所选择的端口。

函数类型：

主动函数

返回值类型：

无返回值

实例：

PortClose(6)

PortInput(*Port, Start, End, Timeout, Length*)

变量	类型	说明
port	int	需读取的原始端口
start	int	起始字符，如果有。
end	int	结束字符，如果有。
timeout	int	内部字符读取时间，如果有。
length	int	最大字符数，如果有。

说明：

从变量 port 指定的端口读取字符串。使用其它变量来控制输入过程，如果 start 不等于 0，当收到 start 设定的字符后，读取过程才开始。如果 start 等于 0，读取过程立即开始。读取过程在下面的其中一个条件满足时结束。

- end 不等于 0 时，收到 end 设定的字符后读取过程结束。
- timeout 不等于 0 时，读取时间超过 timeout 设定的时间后读取过程结束。
- length 不等于 0 时，收到 length 设定的字符数后读取过程结束。

该函数返回已收到的字符串，不包含 start 和 end 字符。它主要在原始通讯口中用 Crimson 编程语言完成自定义协议。它替代了 Edict 中 RYOP 功能。

函数类型：

主动函数

返回值类型：

字符串

实例：

Frame := PortInput(1, '*', 13, 100, 200)

PortPrint(*Port, String*)

变量	类型	说明
port	int	需写出的端口
string	cstring	需传输的字符串

说明：

向 port 指定的通讯口发送 string 中包含的文本。通讯口必须设置使用原始驱动器，如原始串行通讯驱动器，或原始 TCP/IP 驱动器。数据传送后，函数返回。通讯口驱动器按需要控制握手和传输使能线等。

函数类型：

主动函数

返回值类型：

无返回值

实例：

PortPrint(1, "ABCD")

PortRead(*Port, Period*)

变量	类型	说明
port	int	要读取的原始端口
period	int	读取的时间

说明：

从 port 指定的通讯口中读取一个字符。通讯口必须设置使用原始驱动器，如原始串行通讯驱动器，或原始 TCP/IP 驱动器。在设定的时间内如果没有读到数据，则返回数值-1。设定 period 等于 0，会无法将任何数据返回，但可防止 Crimson 因等待数据而花费时间。

函数类型：

主动函数

返回值类型：

整数

实例：

Data := PortRead(1, 100)

PortWrite(*Port, Data*)

变量	类型	说明
port	int	要写出的原始端口
data	int	要传送的字节

说明：

向 port 指定的通讯口发送 data 中的数据。通讯口必须设置使用原始驱动器，如原始串行通讯驱动器，或原始 TCP/IP 驱动器。数据传送后，函数返回。通讯口驱动器按需要控制握手和传输使能线等。

函数类型：

主动函数

返回值类型：

无返回值

实例：

PortWrite(1, 'A')

Power(*Value, Power*)

变量	类型	说明
value	int / float	需处理的数值
power	int / float	Value 所取的幂

说明：

返回 value 的 power 次幂。

函数类型：

被动函数

返回值类型：

整数或浮点数，按变量 value 的类型确定。

实例：

Volume := Power(Length, 3)

Rad2Deg(*Theta*)

变量	类型	说明
theta	float	需计算的角度

说明：

将 theta 从弧度转成角度。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Right := Rad2Deg(Pi()/2)

Random(***Range***)

变量	类型	说明
range	int	随机值的范围

说明：

返回从 0 至 range-1 间的一个随机值。

函数类型：

被动函数

返回值类型：

整数

实例：

Noise := Random(100)

ReadData(*Data, Count*)

变量	类型	说明
data	any	需读取的第一个数组单元
count	int	读取的单元数

说明：

请求在下一个通讯 扫描中读取数组单元 data 起的 count 个单元的数据。该函数通常用在数组被映射为外部数据，并且读取规则设为手动读取时使用。函数立即返回，不等待数据的读取。

函数类型：

主动函数

返回值类型：

无返回值

实例：

ReadData(array1[8], 10)

ReadFileLine(*File*)

变量	类型	说明
file	int	以 openfile 打开，待处理的文件

说明：

返回 file 文件中的一行文本。

函数类型：

主动函数

返回值类型：

字符串

实例：

Text := ReadFileLine(hFile)

Right(*String, Count*)

变量	类型	说明
string	cstring	需处理的字符串
count	int	返回的字符个数

说明：

返回 string 中最后 count 个字符。

函数类型：

被动函数

返回值类型：

字符串

实例：

Local := Right(Phone, 7)

Scale(*Data, R1, R2, E1, E2*)

变量	类型	说明
data	int	需标定的数值
r1	int	Data 中最小原始值
r2	int	Data 中最大原始值
e1	int	与 r1.对应的工程值
e2	int	与 r2.对应的工程值

说明：

此函数用线性方法标定变量 data，假设变量值在 r1 和 r2 间，将在 e1 和 e2.间产生一个相应返回值。中间运算值采用 64 位整数，这样可避免直接使用数学运算符产生的溢出。

函数类型：

被动函数

返回值类型：

整数

实例：

Data := Scale([D100], 0, 4095, 0, 99999)

SendMail(***Rcpt, Subject, Body***)

变量	类型	说明
rcpt	int	数据库地址簿中的收件人索引
subject	cstring	Email 的标题
body	cstring	Email 的内容

说明：

从人机界面中发送一个电子邮件。函数立即返回，将电子邮件加入系统的邮件队列。邮件将发往数据库文件中设定的邮件服务器。

函数类型：

主动函数

返回值类型：

无返回值

实例：

SendMail(1, “Test Subject Line”, “Test Body Text”)

Set(*Tag, Value*)

变量	类型	说明
tag	int / real	需更改的标签
value	int / real	所赋的值

说明：

此函数向指定的标签赋予设定值。它与通常的赋值运算符不同，它将删除标签的所有赋值队列，立即置标签为设定值。它使用在赋值运算符的性能无法完成的应用中。.

函数类型：

主动函数

返回值类型：

无返回值

实例：

Set(Tag1, 100)

SetLanguage(*Code*)

变量	类型	说明
code	int	需选择的语言

说明：

按变量 code 设置界面的当前语言。

函数类型：

主动函数

返回值类型：

无返回值

实例：

SetLanguage(1)

SetNetConfig(*Port, Addr, Mask, Gate*)

变量	类型	说明
port	int	以太网口索引号，必需设为 0
addr	int	网口的 IP 地址
mask	int	网口的掩码
gate	int	网口的缺省网关

说明：

更改数据库文件中以太网口的设置。IP 参数是 32 位整数，可以调用函数 TextToAddr( )获取。注意：设置 3 个 IP 参数为 0，会重置端口设置为数据库文件中的设定；新设置必需在重新上电，才有效。

函数类型：

主动函数

返回值类型：

无返回值

实例：

SetNetConfig(0,0,0,0)

SetNow(*Time*)

变量	类型	说明
time	int	设置的新时间

说明：

按 time 中的整数值设定当前时间。整数值表示从 1997 年 1 月 1 日起到现在的总秒数时间。此数值通常由其它时间/日期函数产生。

函数类型：

主动函数

返回值类型：

无返回值

实例：

SetNow(252288000)

Sgn(*Value*)

变量	类型	说明
value	int / float	待处理的数值

说明：

变量 value 小于 0 时返回-1，大于 0 时返回+1，等于 0 时返回 0。

函数类型：

被动函数

返回值类型：

整数或浮点数，按 value 的类型而定。

实例：

State := Sgn(Level)+1

ShowMenu(*Name*)

变量	类型	说明
name	显示页面	按弹出菜单样式显示的页面

说明：

按弹出菜单的样式显示页面。该函数仅适用于触摸屏。弹出菜单显示在屏幕的上方，左对齐。

函数类型：

主动函数

返回值类型：

无返回值

实例：

ShowMenu(Page2)

ShowPopup(*Name*)

变量	类型	说明
name	显示页面	按弹出窗口样式显示的页面

说明：

按弹出窗口的样式显示页面。弹出窗口显示在屏幕的中央，并处于页面的顶层。窗口可以通过调用 HidPopup()函数退出；也可以通过调用 GotoPage()函数显示新页面来退出；再或者通过定义的按键动作来退出。

函数类型：

主动函数

返回值类型：

无返回值

实例：

ShowPopup(Popup1)

Sin(*Theta*)

变量	类型	说明
theta	float	待处理的角度，弧度为单位

说明：

返回角度 theta 的正弦函数值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

yp := radius*sin(theta)

SirenOn()

变量	类型	说明
none		

说明：

鸣响人机界面内部的蜂鸣器。

函数类型：

主动函数

返回值类型：

无返回值

实例：

SirenOn()

Sleep(*Period*)

变量	类型	说明
period	int	暂停的时间，毫秒为单位

说明：

按设定的时间暂停执行当前任务。此函数通常在程序中调用并在后台运行，或在通讯协议中使用。建议不要使用触发器和按键调用此函数。

函数类型：

主动函数

返回值类型：

无返回值

实例：

Sleep(100)

Sqrt(*Value*)

变量	类型	说明
value	int / float	待处理的数据

说明：

返回 value 的平方根值。

函数类型：

被动函数

返回值类型：

整数或浮点数，按 value 值的类型而定。

实例：

Flow := Const * Sqrt(Input)

StdDev(*Element, Count*)

变量	类型	说明
element	int / float	需处理的第一个数组单元
count	int	需处理的单元数.

说明：

假设数组代表所调查人口的采样，此函数会返回数组中 element 单元为起始的，count 个数组单元中数据的标准偏差。如需要计算整个人口的标准偏差，使用函数 PopDev。

函数类型：

被动函数

返回值类型：

浮点值

实例：

Dev := StdDev(Data[0], 10)

StopSystem()

变量	类型	说明
none		

说明：

停止人机界面的操作，使用户可以更新界面中的数据库文件。当界面的编程口用作串行通讯口时，调用该函数可停止所有的通讯，并将通讯口恢复为编程口。

函数类型：

主动函数

返回值类型：

无返回值

实例：

StopSystem()

Strip(*Text, Target*)

变量	类型	说明
text	cstring	需处理的字符串
target	int	要剔除的字符

说明：

从字符串文本中剔除所有设定字符。

函数类型：

被动函数

返回值类型：

字符串

实例：

Text := Strip(“Mississippi”, ’s’)

Text now contains “Miiippi”.

Sum(*Element, Count*)

变量	类型	说明
element	int / float	需处理的第一个数组单元
count	int	需处理的单元数

说明：

返回从 element 单元起始的，count 个单元中数据的总和。

函数类型：

被动函数

返回值类型：

整数或浮点数，按数组类型而定。

实例：

Total := Sum(Data[0], 10)

Tan(*Theta*)

变量	类型	说明
theta	float	待计算的角度，弧度为单位

说明：

返回角度 theta 的正切函数值。

函数类型：

被动函数

返回值类型：

浮点数

实例：

yp := xp * tan(theta)

TextToAddr(*Addr*)

变量	类型	说明
addr	cstring	点-数字形式的地址字符串

说明：

将点-数字形式的地址字符串转换为 32 位 IP 地址

函数类型：

被动函数

返回值类型：

整数

实例：

ip := TextToAddr(“192.168.0.1”)

TextToFloat(*String*)

变量	类型	说明
string	cstring	需处理的字符串

说明：

返回字符串 string 中的数值，数值是浮点数形式。此函数通常与 Mid 函数一起使用，从原始通信口提取字符串中的数值。也可以用来将字符串数值转换为浮点数。

函数类型：

被动函数

返回值类型：

浮点数

实例：

Data := TextToFloat("3.142")

TextToInt(*String, Radix*)

变量	类型	说明
string	cstring	需处理的字符串
radix	int	使用的进制

说明：

返回字符串 string 中的数值，数值是整数形式。此函数通常与 Mid 函数一起使用，从原始通信口提取字符串中的数值。也可以用来将字符串数值转换为整数。

函数类型：

被动函数

返回值类型：

整数

实例：

Data := TextToInt("1234", 10)

Time(*H, M, S*)

变量	类型	说明
h	int	小时，从 0 到 23.
m	int	分钟，从 0 到 59.
s	int	秒钟，从 0 到 59.

说明：

返回从午夜到指定时刻以秒记的时间。此数值可用于其它时间/日期函数。也可与 Date 函数的返回值相加，产生一个特定的日期/时间值。

函数类型：

被动函数

返回值类型：

整数

实例：

t := Date(2000,12,31) + Time(12,30,0)

WaitData(*Data, Count, Time*)

变量	类型	说明
data	any	读取的第一个数组单元
count	int	读取的单元数
time	int	计时时间，毫秒为单位

说明：

请求在下一个通讯 扫描中读取数组单元 data 起的 count 个单元的数据。该函数通常用在数组被映射为外部数据，并且读取规则设为手动读取时使用。与 ReadData()函数不同，此函数将等待一段时间以读取数据，在时间范围内读取完成，返回 1，否则返回 0。

函数类型：

主动函数

返回值类型：

整数

实例：

```
status := WaitData(array1[8], 10, 1000)
```

WriteFileLine(*File, Text*)

变量	类型	说明
file	int	用 OpenFile 函数打开，待处理的文件
text	cstring	写入文件的文本

说明：

向指定文件写入一个字符串，并返回成功写入的字节数。字符串包括回车符，和行缩进字符等。

函数类型：

主动函数

返回值类型：

整数

实例：

count := WriteFileLine(hFile, “Writing text to file.”)

--- end ---